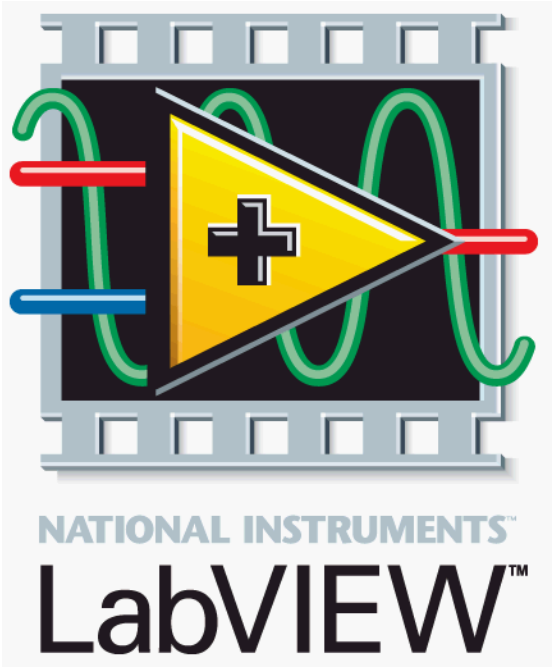
Computer Interfaces for Measurement and Automation in Experimental Investigations

Introduction to LabVIEW Programming Language

Johra, Hicham

Computer Interfaces for Measurement and Automation in Experimental Investigations: Introduction to LabVIEW Programming Language

Hicham Johra



ISSN 1901-726X
DCE Technical Report No 258

DEPARTMENT OF CIVIL ENGINEERING
AALBORG UNIVERSITY

Aalborg University
Department of Civil Engineering
Architectural Engineering

DCE Technical Report No. 258

Computer Interfaces for Measurement and Automation in Experimental Investigations: Introduction to LabVIEW Programming Language

by

Hicham Johra

November 2018

Scientific Publications at the Department of Civil Engineering

Technical Reports are published for timely dissemination of research results and scientific work carried out at the Department of Civil Engineering (DCE) at Aalborg University. This medium allows publication of more detailed explanations and results than typically allowed in scientific journals.

Technical Memoranda are produced to enable the preliminary dissemination of scientific work by the personnel of the DCE where such release is deemed to be appropriate. Documents of this kind may be incomplete or temporary versions of papers—or part of continuing work. This should be kept in mind when references are given to publications of this kind.

Contract Reports are produced to report scientific work carried out under contract. Publications of this kind contain confidential matter and are reserved for the sponsors and the DCE. Therefore, Contract Reports are generally not available for public circulation.

Lecture Notes contain material produced by the lecturers at the DCE for educational purposes. This may be scientific notes, lecture books, example problems or manuals for laboratory work, or computer programs developed at the DCE.

Theses are monograms or collections of papers published to report the scientific work carried out at the DCE to obtain a degree as either PhD or Doctor of Technology. The thesis is publicly available after the defence of the degree.

Latest News is published to enable rapid communication of information about scientific work carried out at the DCE. This includes the status of research projects, developments in the laboratories, information about collaborative work and recent research results.

Published 2018 by
Aalborg University
Department of Civil Engineering
Thomas Manns Vej 23
DK-9220 Aalborg Ø, Denmark

Printed in Aalborg at Aalborg University

ISSN 1901-726X
DCE Technical Report No. 258

Contents

Introduction

For decades, the experimental investigations in the field of building energy and indoor environment were mostly based on tests and measurements with steady state conditions. The use of simple conventional electronic controllers was then sufficient to provide good results and tackle the scientific problems of this time.

However, since the years 2000, the building energy and indoor environment industry includes more and more dynamic control strategies such as temperature set point schedules, night time set back, adaptive ventilation, building energy flexibility strategies, etc. The building's HVAC systems used to be controlled independently from each other, often leading to sub-optimal energy performance and poor indoor comfort. Nowadays, the building energy management strategies tend to control all HVAC systems together and include advanced controllers such as model predictive control to optimize both energy performance and indoor comfort taking into account the building thermodynamics, weather forecast, users behavior, energy price and building energy flexibility potential.

This paradigm shift obliges research teams to test HVAC systems and building elements with complex transient conditions and all sorts of dynamic strategies. It is therefore no longer possible to have laboratories with only independent electronic controllers and stand-alone instruments. The computer tools must be efficiently used to control all the components of the experimental setup, centralize the measurement data acquired from the different instruments, run complex control algorithms and distribute control signals to actuators while providing clear feedbacks to the users.

The LabVIEW programming language has been specifically developed to allow any laboratories to create their own computer interface to control most, if not all, their devices. This graphical programming language is ideal to make well-behaved interfaces and applications with clear architectures which are easy to maintain and upgrade. The LabVIEW programming language is widely used by dozens of thousands of people all around the world and in many laboratories and companies. The strength of this active community and the quality of the technical support provided by *National Instruments* make LabVIEW the perfect tool to develop flexible computer interfaces and applications for the laboratories of the Department of Civil Engineering of Aalborg University.

The aim of this document is to provide a basic introduction to the LabVIEW programming language. Some simple exercises are presented to help understanding the basics about the LabVIEW programming language, and to be able to acquire an analogue signal from a sensor with a simple National Instruments data acquisition board, to communicate with a more complex digital instrument, and to program a basic control system. Because LabVIEW is a graphical programming environment, it is also a great tool to comprehend the basics of algorithmic and programming for automated measurements and graphical user interfaces.

Equipment overview

Before starting the introduction exercises, please make sure that all equipment is present in the dedicated box. If something is missing or is broken, please contact Hicham Johra: hj@civil.aau.dk.

The following equipment should be found in the dedicated storage box (see *Figure 1*):

- A USB memory stick with all the necessary documents and exercises. A copy of all documents and exercises can be found on Moodle.
- A printed version of this document.
- 2 laptops "Labview PC 1" and "Labview PC 2" with their respective power supplies (**1**).
- 2 USB computer mice (**2**).
- 2 NI USB-6009 (National Instruments USB data acquisition boards) with their dedicated USB cables (**3**).
- 2 electronic thermometers (**4**).
- Printed version of the technical documentation of the electronic thermometers (**5**).
- 2 USB to serial converters (blue colour) (**6**).
- 3 thermocouple amplifiers (PM 9874 A TC linearizer PHILIPS) with 1 thermocouple mounted on each + their respective power supply cables and analogue signal output cables (**7**).
- 2 little controlled boxes with a small fan and a small lamp bulb mounted on them (**8**).

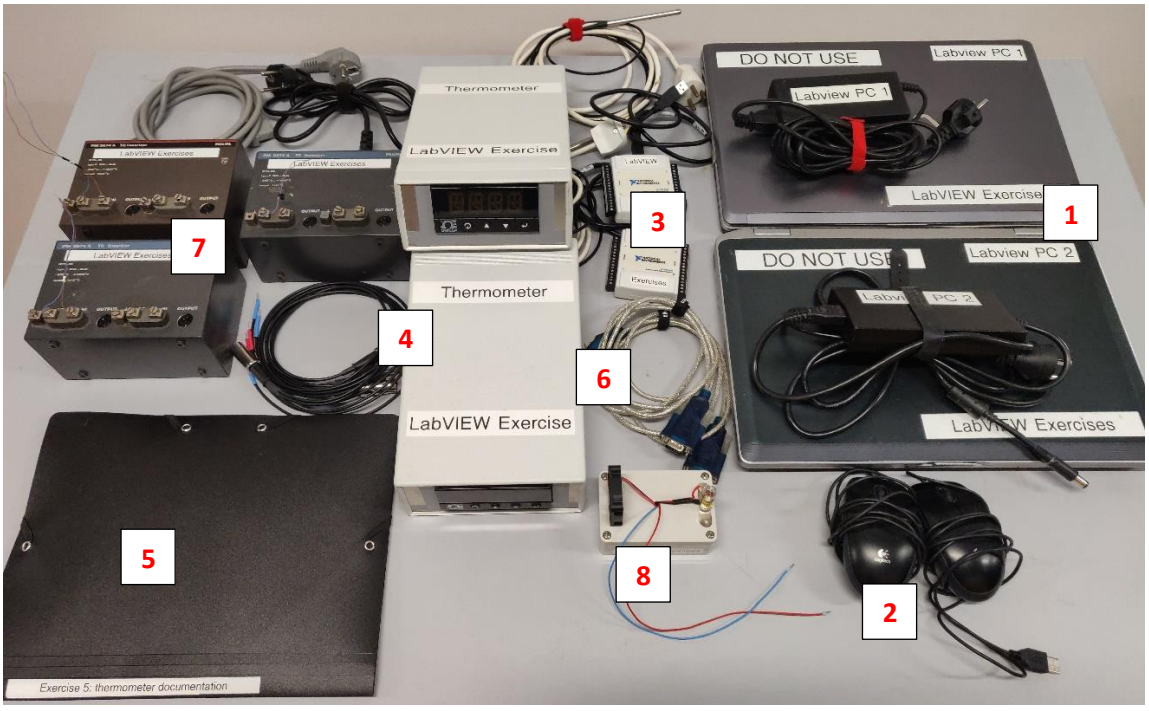


Figure 1: *Overview of all equipment needed for the LabVIEW introduction exercises.*

Programming with LabVIEW

LabVIEW is a graphical programming environment which allows the creation of all sorts of application program and graphical user interface. It is widely used for research and industrial processes, particularly for automation, system control, machine interface and measurements.

A LabVIEW program is called a "VI" (Virtual Instrument). A VI a composed of a "front panel" and a "block diagram" (see *Figure 2*). The front panel is what is visible by the user on the screen when the VI is running. The front panel is used to get inputs from the users (button, input parameters) and to display information to the users (graphs, indicators). The block diagram is where the code actually is. The block diagram contains all the different block functions, linked together in a certain way so that they perform a specific task.

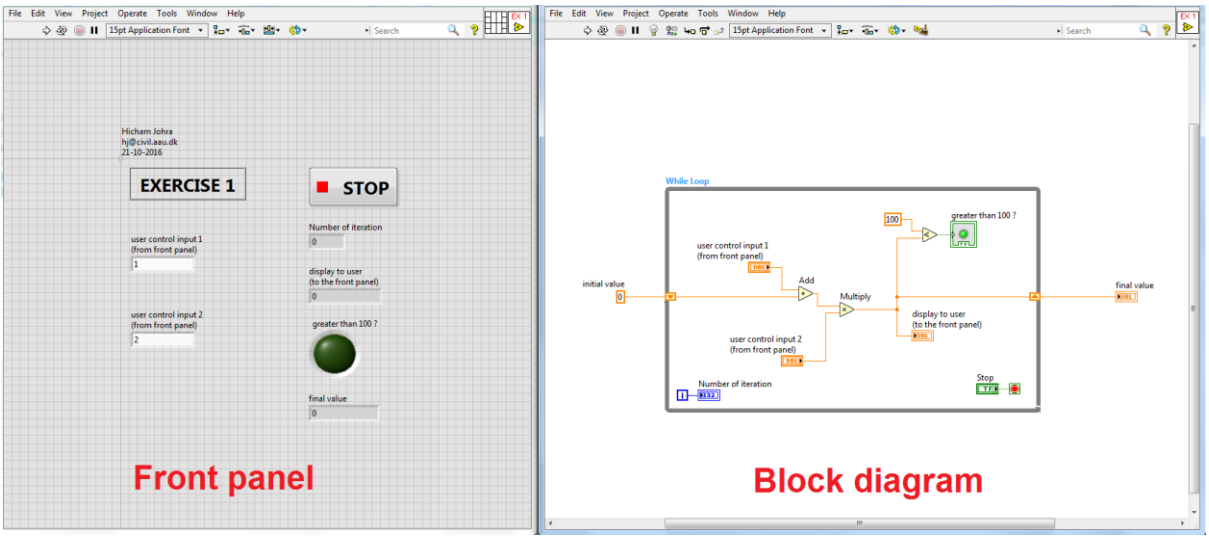


Figure 2: *VI front panel (left) and block diagram (right).*

LabVIEW programming is based on the **data flow programming** principle. Similarly to Simulink (MATLAB), a function is represented by a block. The information flows from the output pins of a function block to the input pins of other function blocks.

Usually, the information / data flow is from left to right. The inputs of a function block are placed on the left side of the block and the outputs are placed on the right side of the block (see *Figure 3*).

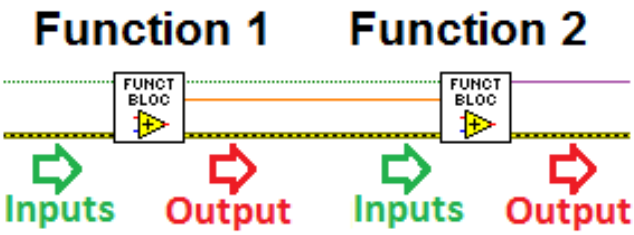


Figure 3: *Data flow.*

When the “RUN” button is pressed (see *Figure 4*), the function block will be executed if and only if they received information from all their respective input pins. Therefore the function blocks can be executed sequentially in a definite order (see *Figure 5*).

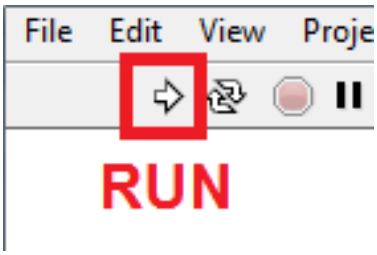


Figure 4: *Run button.*

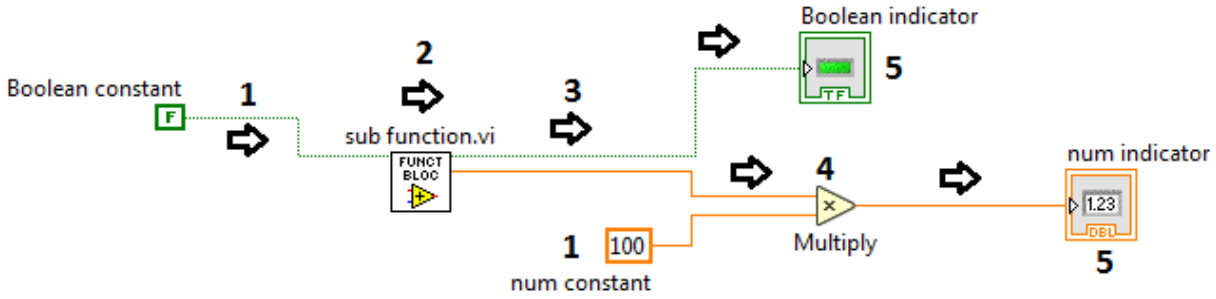
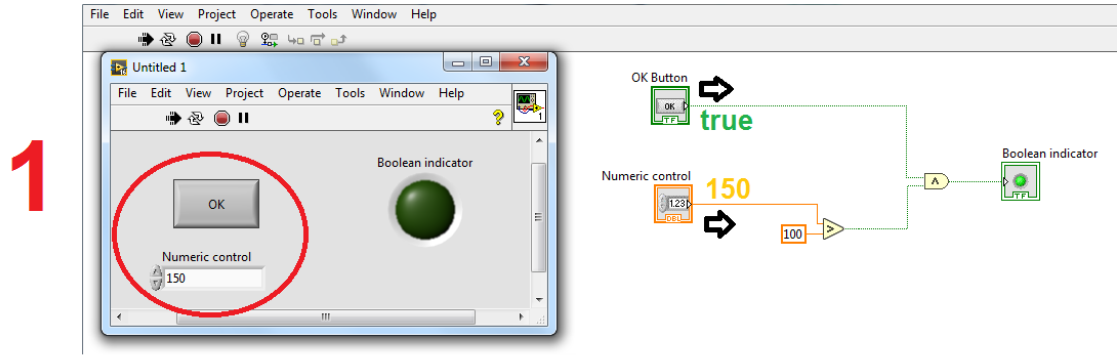
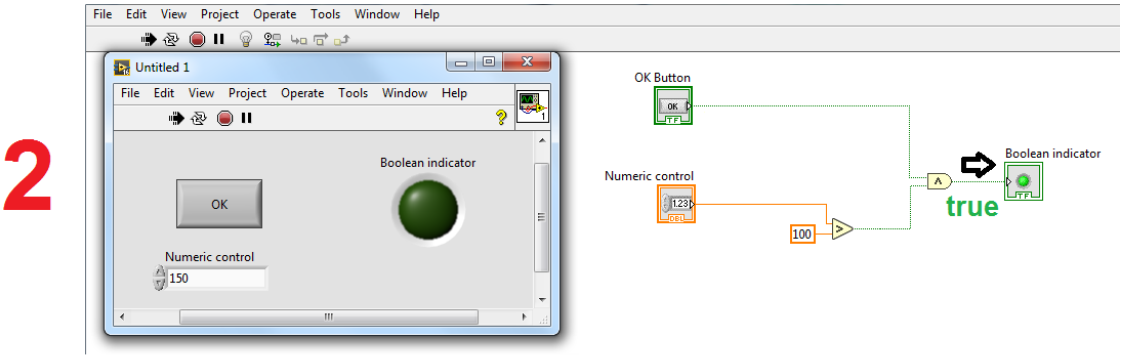


Figure 5: *Data flow.*

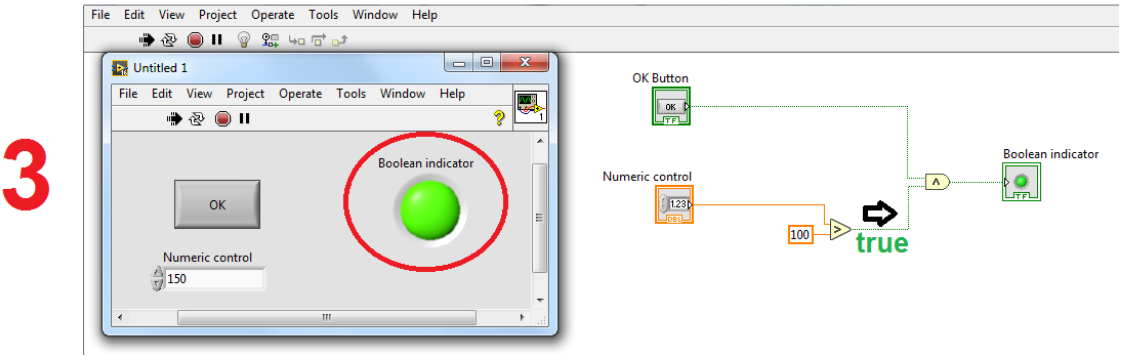
When the user interacts with the interface via the front panel by pressing buttons, the associated functions on the block diagram are activated and send signals. The example below (see *Figure 6*) show a simple block diagram where the user presses the “OK” button. The data flows until it reaches the “Boolean” (true or false) indicator. The input parameter “number” can also be changed by the user from the front panel.



The user pushes the "OK" button and set a certain value to the "Numerical control".



Data flow from left to right, from block functions to block functions.



The boolean indicator receives the true value and lights on the associated LED on the front panel.

Figure 6: *Step by step execution of a simple VI.*

In LabVIEW, there are different types of data going through the wires from block functions to block functions. The most common data types have a specific wire color: integer, floating point numeric, string (text), boolean (true or false), error (an error message is a signal send by a function which cannot be executed correctly in order to warn the user that something wrong is going on)… They can be transmitted in wires as a scalar (1 single value at a time) or in a list / 1D array / vector / group or in a matrix / 2D array or higher dimension array (see *Figure 7*).

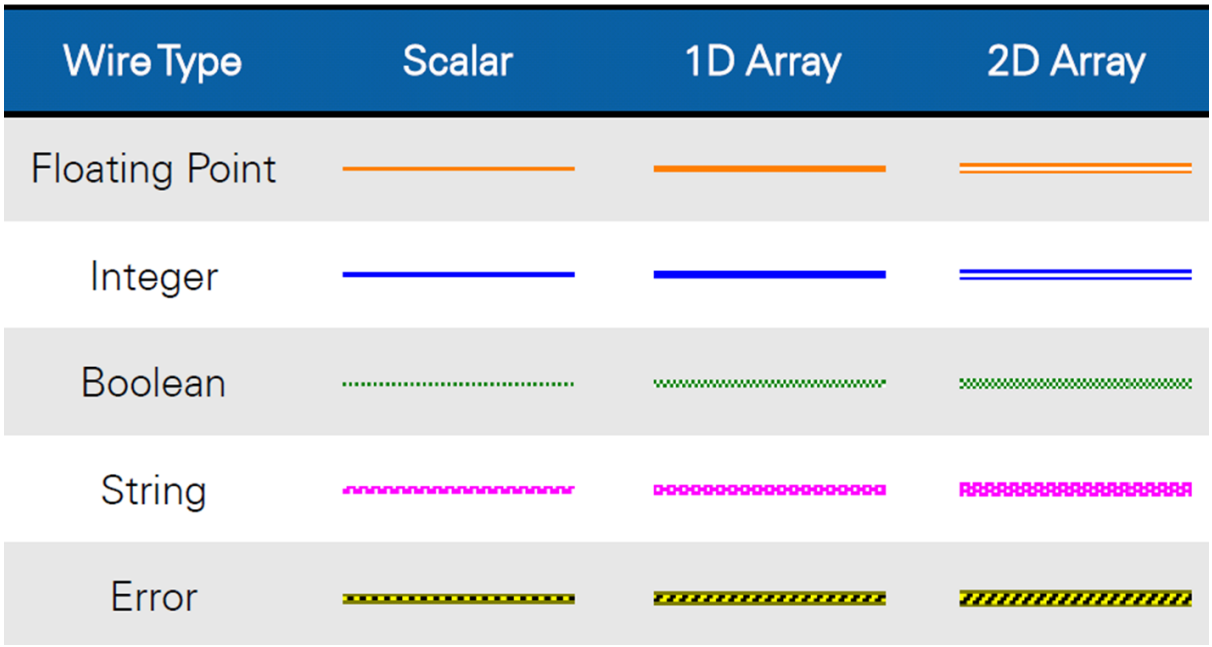


Figure 7: *Some examples of different types of data in LabVIEW.*

You can hover over the pins of the function's block with the cursor to see what are the names of the input pins and output pins to be connected.

Right click on the block diagram or on the front panel to access the different libraries and function palettes to implement all possible functionalities (see *Figure 8*).

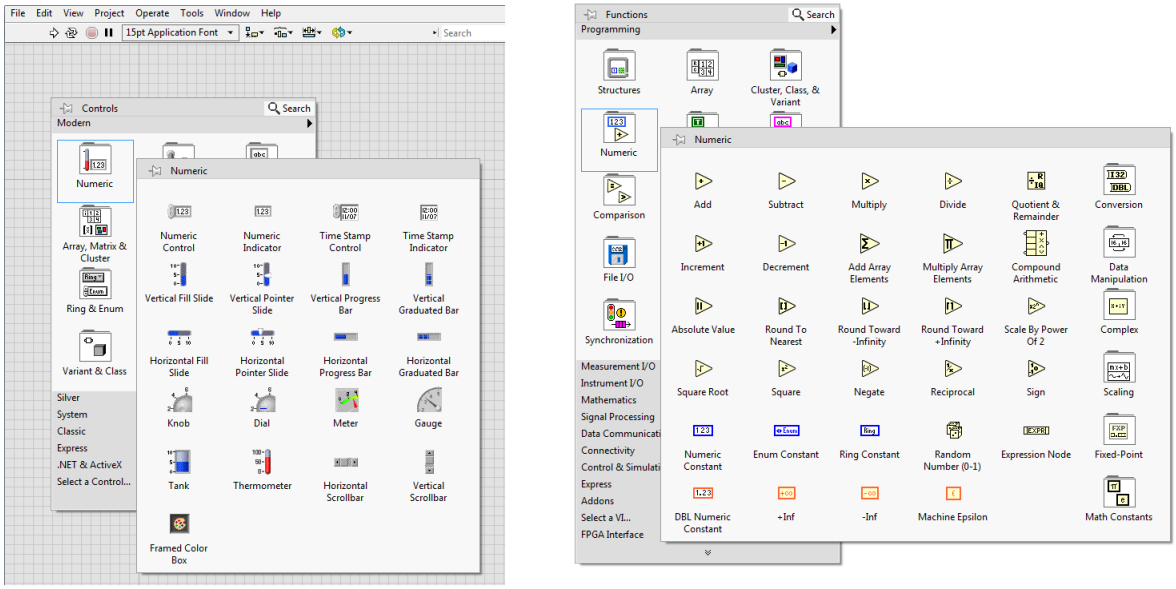


Figure 8: *Right click on front panel and block diagram to access palettes.*

Right click on a data constant, data wire, block pin or block function to access the associated function palette or create a constant, an indicator (to show a something to the user on the front panel) or a control (to get input from the user on the front panel) associated to this data type (see *Figure 9*).

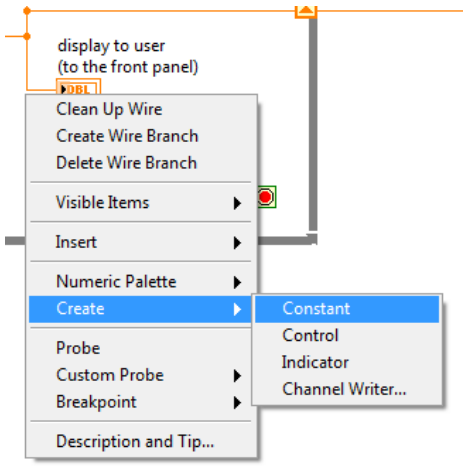


Figure 9: *Right click on data type and add constant, indicator or control.*

Use the “Ctrl” + “space” to find a function by typing its name. “Right click” -> “Help” on a block function to access its description and help file. Alternatively, use “Ctrl” + “h” to enable/disable the “Context Help” window.

Similarly to “classical” text programming languages, LabVIEW has all the basic structures such as “for loops”, “while loops” (see *Figure 10*), if condition, case selector (see *Figure 11*), etc.

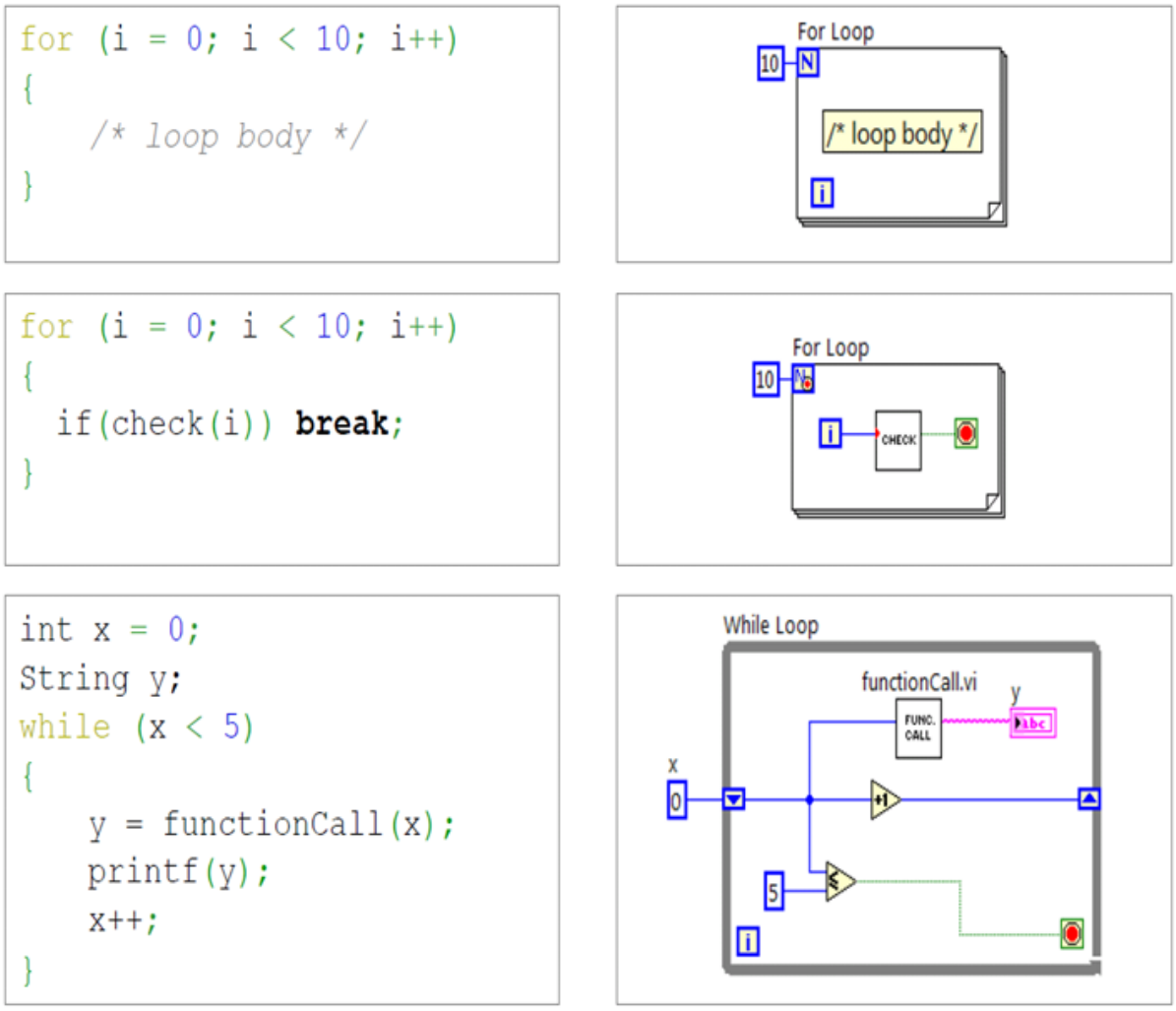
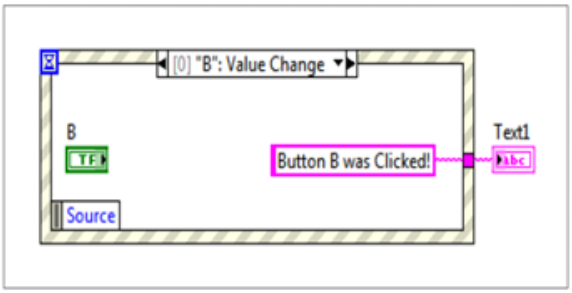


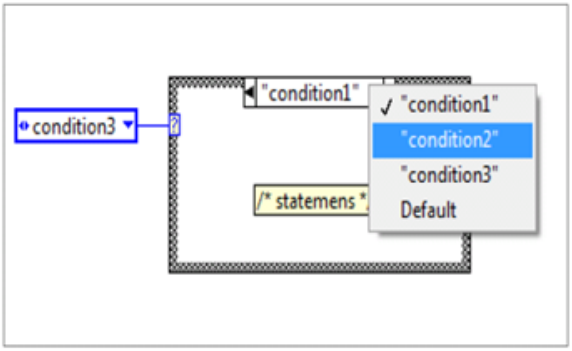
Figure 10: *Different loop structures.*

```
Button B = new Button();
B.Click += new RoutedEventHandler(OnBClick);

void OnBClick(object Source)
{
      Text1.Text = "Button B was Clicked!";
}
```




```
if condition1 then
  -- statements;
elseif condition2 then
  -- more statements
elseif condition3 then
  -- more statements;
else
  -- other statements;
end if
```




```
switch (n) {
  case 5:
    printf("Small number.");
    break;
  case 100:
    printf("Large number.");
    break;
default:
    printf("Outside range");
    break;
}
```

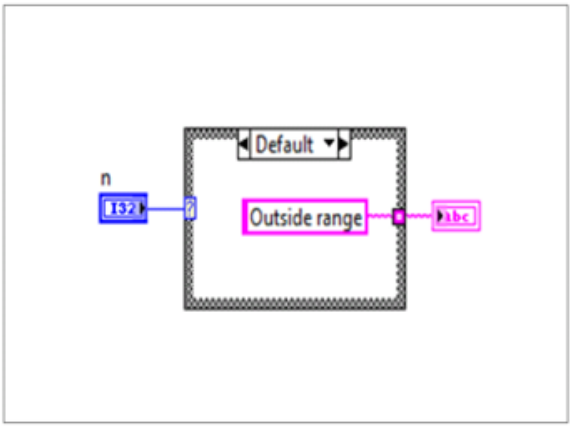


Figure 11: *Different case and if conditions structures.*

Many examples, which are ready to be used, are freely available from the LabVIEW database (see *Figure 12*).

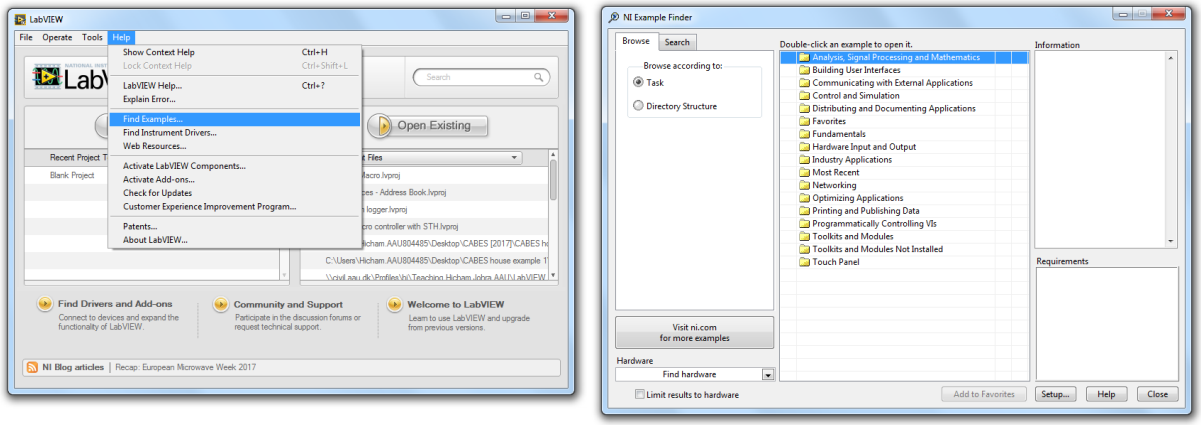


Figure 12: *Find VI examples.*

After some practice, you will be able to create well-structured interfaces with state-or-the-art architectures. Most of the interfaces have similar features. They start with an initialization sub-function, then run several loops in parallel to perform separate tasks at the same time (get input from front panel, manage information and tasks, read the instrument, treat the data, log the data in a log file, display the data on the front panel…), and end with a termination sub-function (see *Figure 13*).

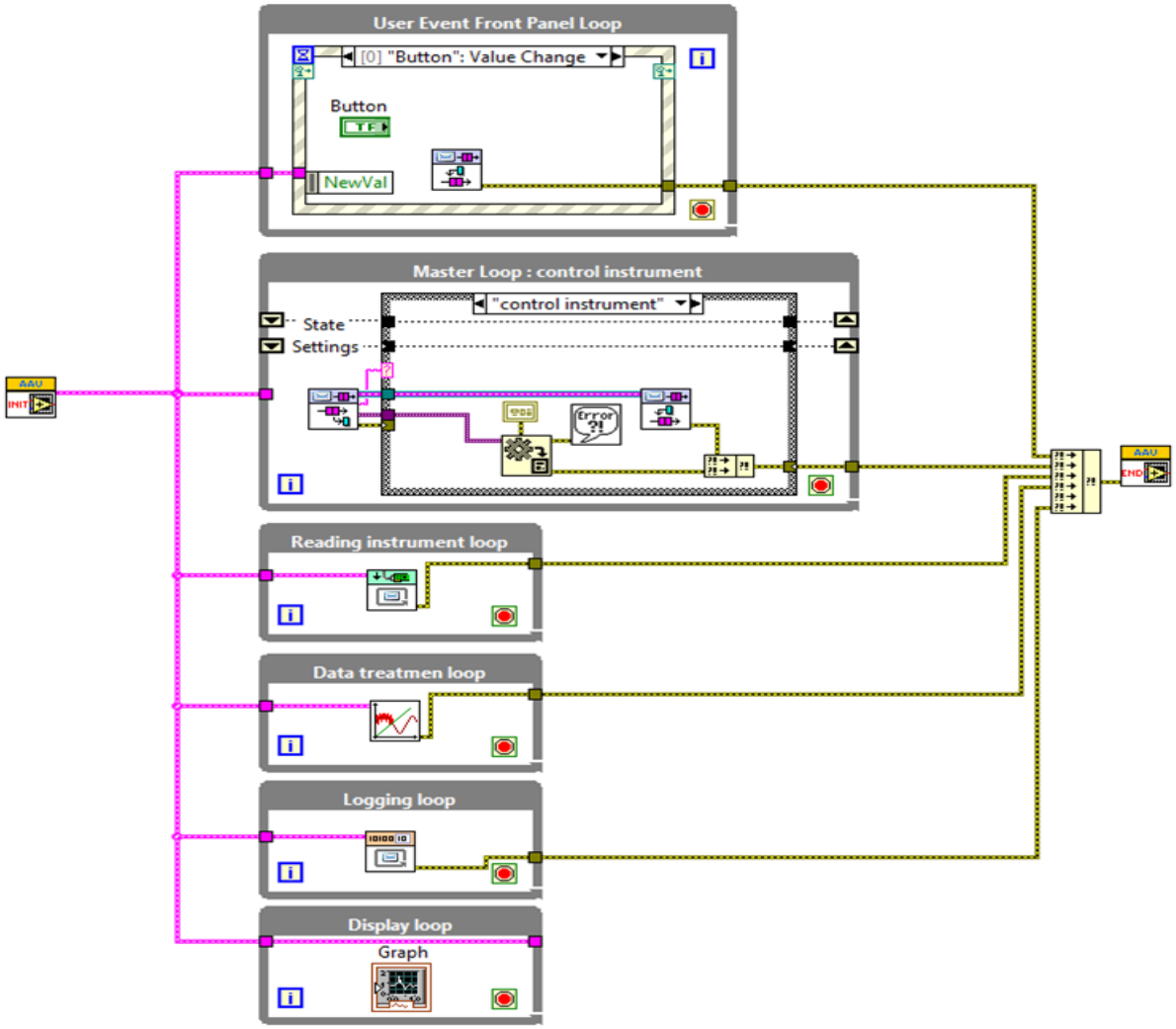


Figure 13: *Block diagram of an interface with the "message handler / state machine" architecture.*

Don’t forget to google it if you want some help or some examples. LabVIEW has a large and active community on the Internet and you can easily find the answer to all your questions on the programming forums or on the official website of National Instruments.

You can download LabVIEW and access online courses for free by creating an account with your AAU email address:

http://www.ni.com

Download the last version of LabVIEW (select “existing users” and download the 32-bit version of LabVIEW):

http://www.ni.com/da-dk/shop/labview/download.html

For more information about LabVIEW programming, please read the following tutorial:

https://www.ni.com/getting-started/labview-basics/

For taking free online courses, go on:

http://www.ni.com/self-paced-training/

Some tips and tricks

- Use “Ctrl” + “e” to open the block diagram.
- If you do not know which function to use: just google it: for example “get mean value from vector labview”.
- Try to keep your block diagram clean and tidy. Place the functions and the wires in a logical way in the order they should be executed from left (for the first one) to right (for the last one). You can select a function block (left click) or a wire leg (left click) and move it with the mouse or with the arrow buttons. Double click on a wire to select it entirely (all legs). Right click on a wire -> “Clean Up Wire” to automatically reshape the wire in a clearer way.
- Double click on a wire to select it entirely and press “Delete” button to delete it.
- Use “Ctrl” + “b” to delete all broken wires (dotted black wires) from the block diagram.
- When you do not know what a function is doing on the block diagram, right click on it -> “Help” to figure it out. Alternatively, use the context help window (“Ctrl” + “h”).
- Hover over the pins of the block function with the cursor to see what is the name of the input pins and output pins to be connected. Right click on these pins -> “Create” to add a constant, an indicator (to show result on front panel) or a control (to change value during execution on the front panel) associated to the data type of this pin. If you don’t know what input to place for a certain function, try to add a constant by right clicking on it, it will create a constant of the correct data type with the default value for this block function.
- Use vectors (array) to store list of values, numbers, strings, Booleans. Use the index function to access a specific element in this vector.
- Remember to initialize values and data vector before entering a loop (from the left). Create a shift register to pass the updated version of the data vector from one iteration of the loop to the other. When the loop is stopped, get the output (last update of the data vector) from the right output pin of shit register on the right side of the loop. Place your post data treatment on the right side of the loop.

Exercise 1

The goal is this first very simple example is just to have a basic understanding of how the data flow is working in LabVIEW.

Open the LabVIEW VI shortcut on the computer desktop named “Exercise 1”.

The front panel of the VI is open (see *Figure 14*).

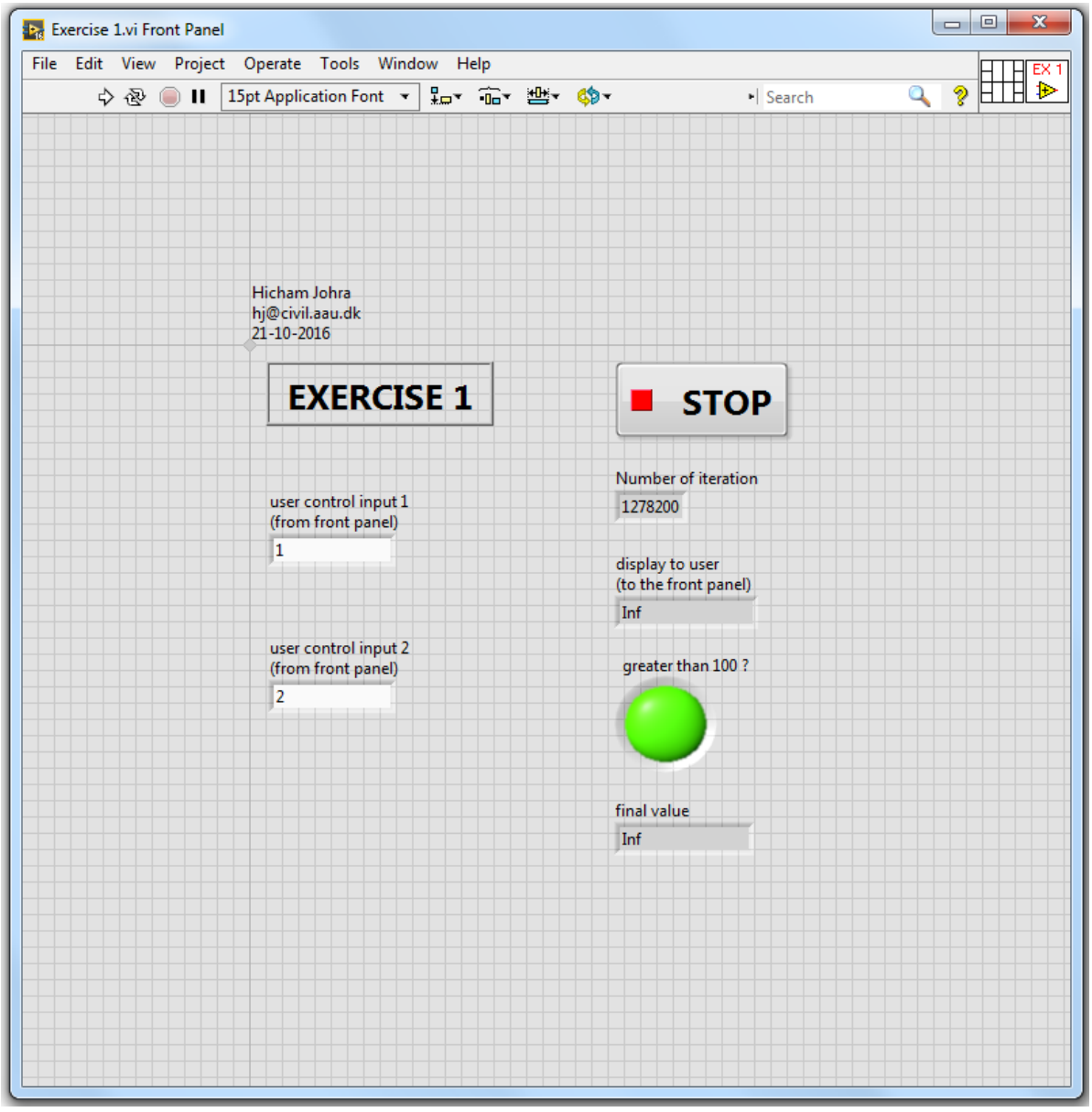


Figure 14: *VI front panel.*

Access the block diagram (see *Figure 15*) by pressing “Ctrl + E”.

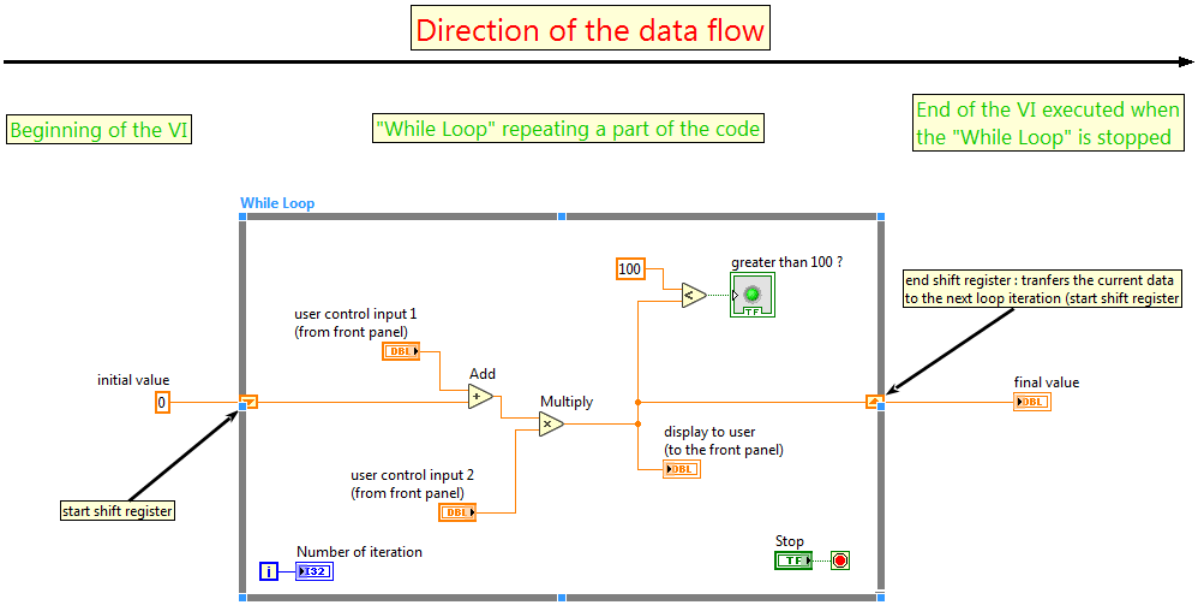


Figure 15: *VI block diagram.*

Press “highlight” button (see *Figure 16*) in order to slow the computation down to step by step speed. Therefore you can follow on the block diagram what is going on, where is going the data.

Press the “RUN” button (see *Figure 16*) and check what is happening on the block diagram and on the front panel (see *Figure 17*).

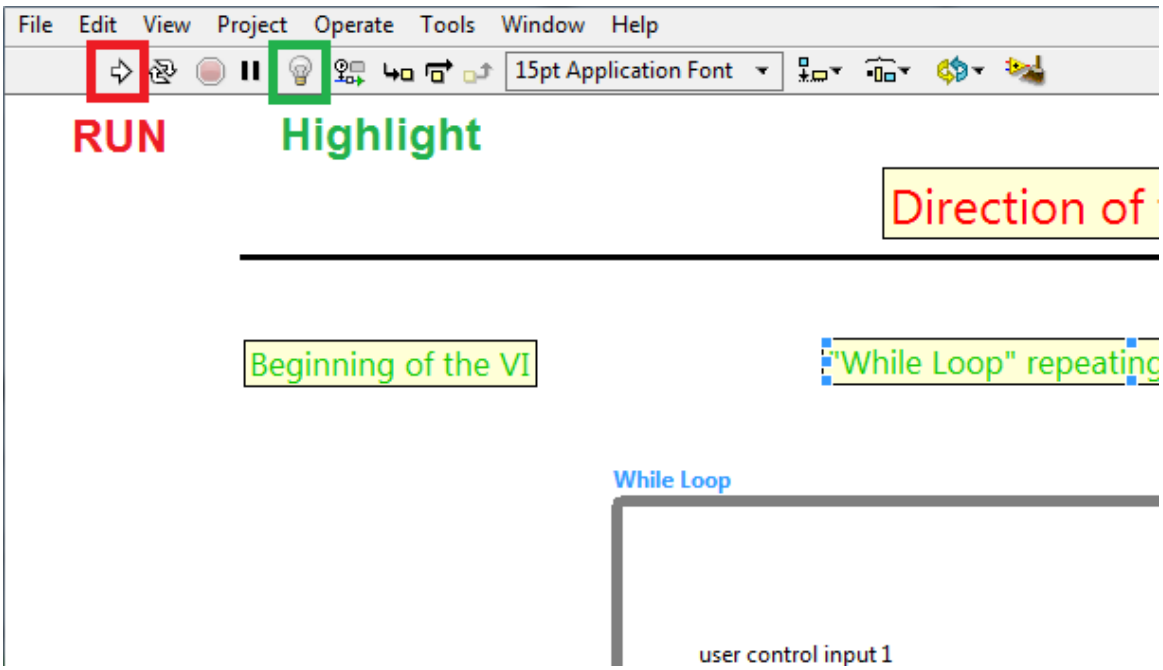


Figure 16: *VI execution bar.*

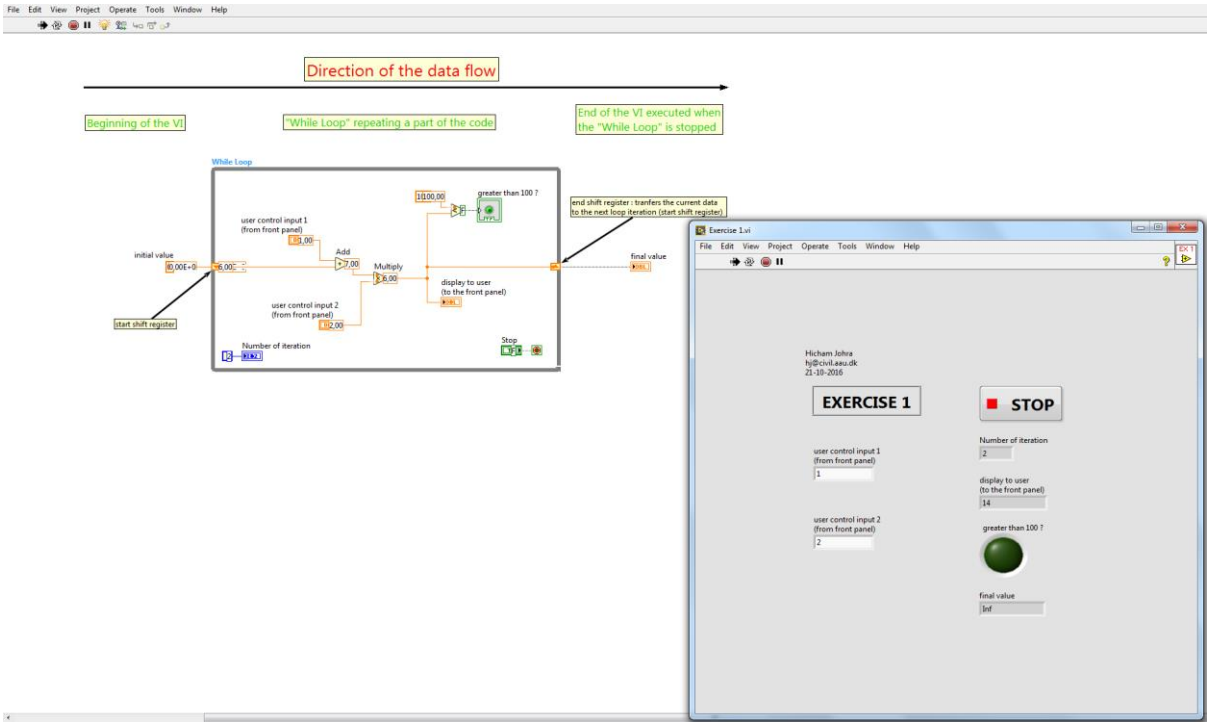


Figure 17: *VI running step by step.*

The data flow from the left to the right of the diagram and each function is executed in a certain order.

Part of the code is encapsulated inside a “while loop”. This structure repeats the part of code inside it until the “stop” signal is taking the value “true” (see *Figure 18*).

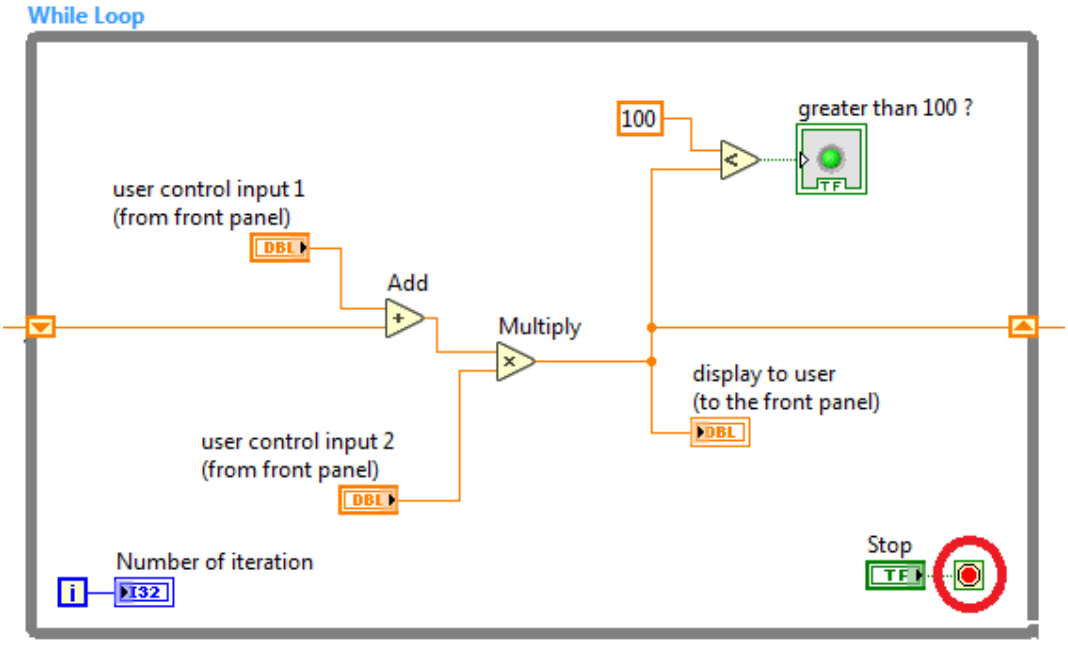


Figure 18: *Stop condition terminal of the while loop.*

In order to “save” the information of the current iteration and pass it over to the next iteration, the “shift register” structure is used. The shift register is placed at the beginning (left side) and at the end (right side) of a loop. When the data arrives to the end side of the shift register, it is passed over to the beginning side of the shift register for the next iteration of the sub-diagram contained by the loop (see *Figure 19*).

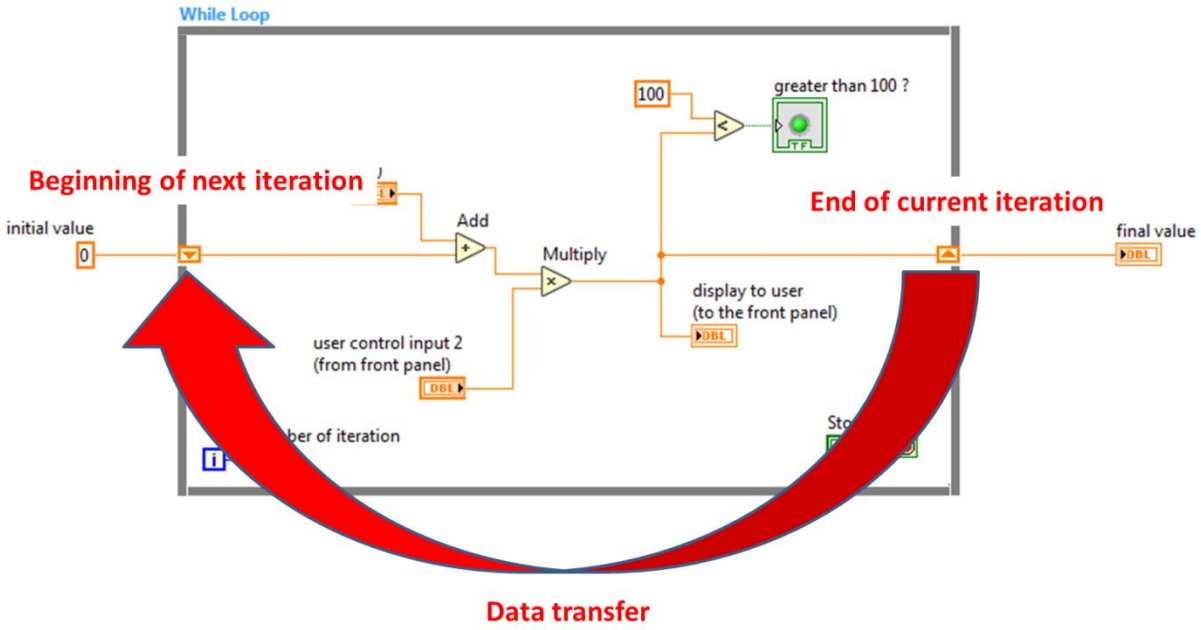


Figure 19: *Data flow with shift register in a while loop.*

Press the “Stop” button on the front panel in order to stop the VI. The “stop” button function’s block will send a “true” Boolean value to the stop condition terminal of the while loop which will stop the execution of the loop and terminate the whole VI.

Close the VI without saving.

Exercise 2

The goal of this second exercise is to create a program from scratch, which will perform a given task (described hereafter).

Open LabVIEW, create a new VI and save it: “File” -> “New VI” (see *Figure 20*).

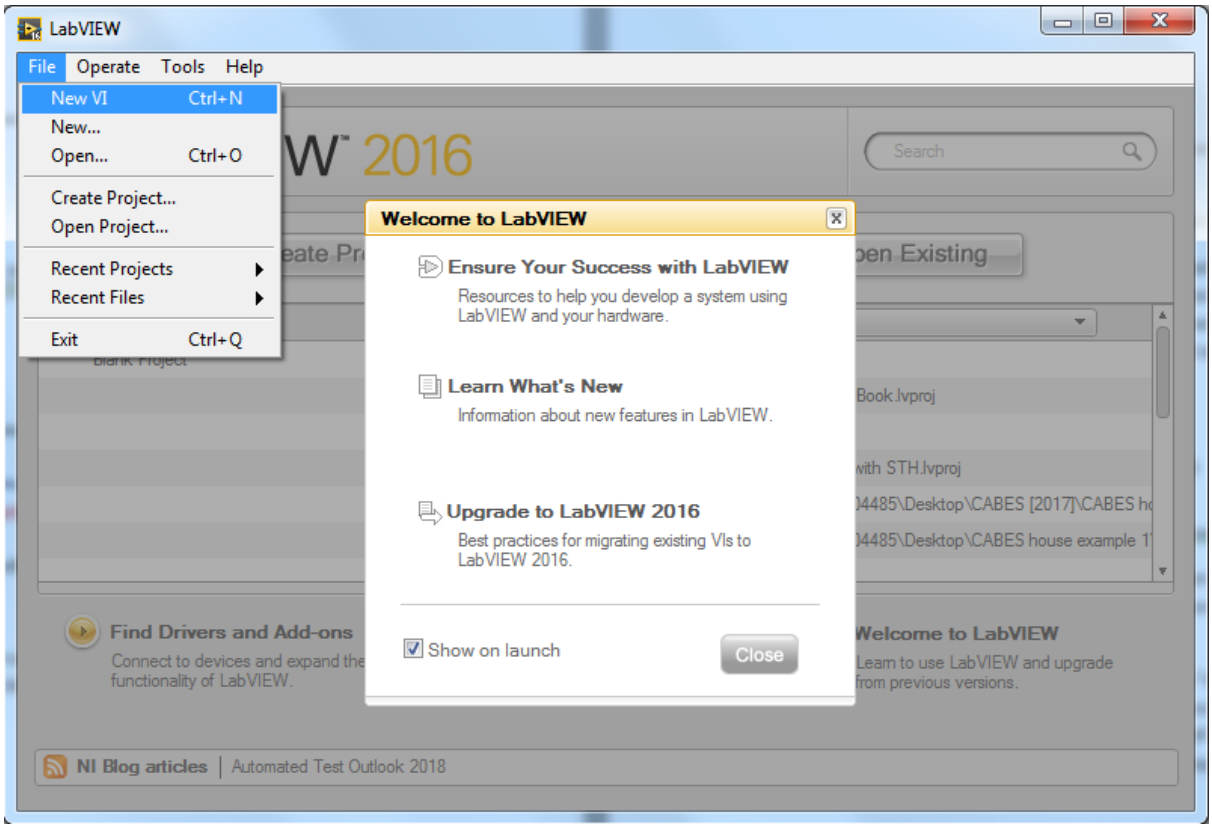


Figure 20: *Create a new VI.*

Objectives

Create your own LabVIEW code which can perform the following task:

Give the first term of the Fibonacci sequence which is above a certain limit (input from the user on the front panel). Once this last term is calculated, the length of this Fibonacci sequence (how many terms have been calculated until the limit) is shown together with the sum of all the calculated terms. These results are displayed on the front panel with numerical indicators and with a text string indicator. A speedometer-like indicator shows the value of the current term being calculated (see *Figure 21*).

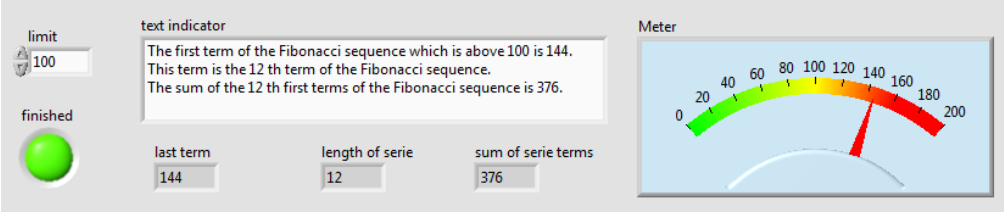


Figure 21: *Front panel of the solution after the calculation is finished.*

In the example (see *Figure 21*), the limit is set to 100 by the user. The Fibonacci sequence is calculated until one term is above 100. Here it is the case for the term which is equal to 144. The iterative calculation of the Fibonacci sequence is thus stopped and the results "144" is displayed on the "last term" numerical indicator and the "meter" indicator, the "finished" Boolean indicator is set "true", the length of the sequence and the sum of all the terms until this limit are displayed on numerical indicators. A text string indicator gives information about the result.

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \ldots$$

Figure 22: *The 12 first terms of the Fibonacci sequence.*

This task is not very useful for any engineering purpose, it is just an exercise for you to learn how to perform iterative calculation with LabVIEW programming.

Help

The calculation of the Fibonacci sequence is performed according to the following algorithm:

$$F_{n+1} = F_n + F_{n-1}$$

With the initial conditions as:

$$F_0 = F_1 = 1$$

In simple words: the first two terms of the Fibonacci sequence are 1. The next term of the sequence is the sum of the two previous ones.

The calculation should be performed iteratively inside a “while loop” structure. This means that the part of the block diagram doing the iterative calculation should be placed inside a while loop. You can create a while loop with the structure palette by right-click on the block diagram (see *Figure 23*).

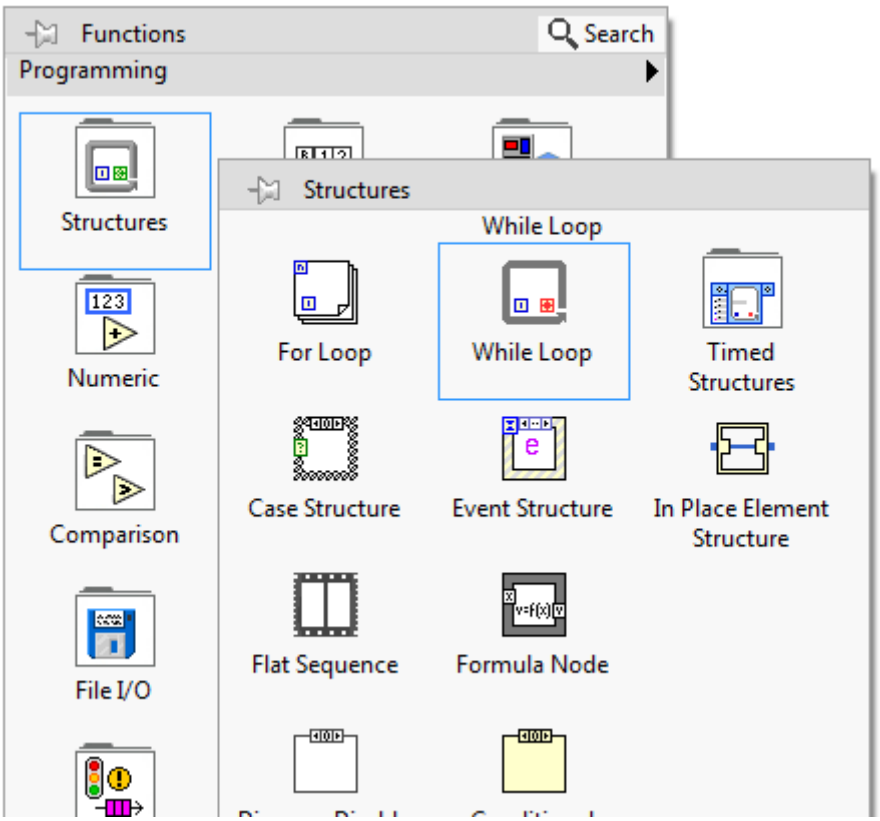


Figure 23: *While loop in the structure palette.*

The while loop stops when the last calculated term of the Fibonacci sequence exceeds the limit (set as input from the front panel with a “control” block function). The end / stop signal is set true and sent to the stop condition terminal of the while loop and shown on the front panel with a boolean indicator(see *Figure 24*).

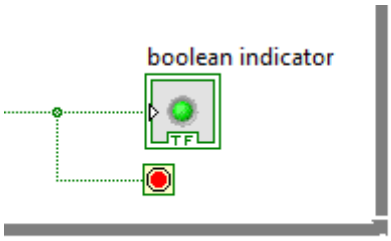


Figure 24: *Stop condition terminal of the while loop with Boolean indicator*

Each iteration of the while loop should last at least 100 ms: place a “wait” function (see *Figure 25*) inside the while loop.

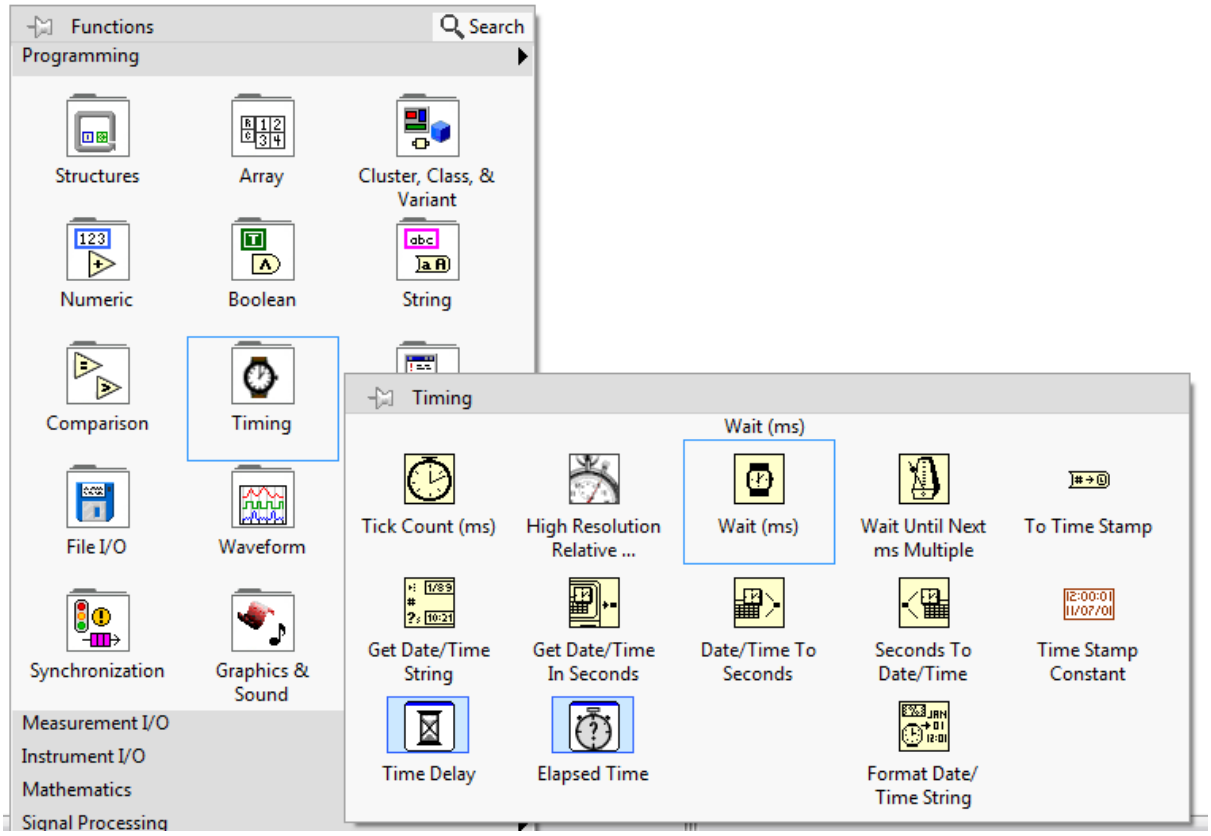


Figure 25: *Wait function in the timing palette.*

In order to store the consecutive terms of the sequence, they should be put in a list or array (1D array). Array can be created and handled with functions from the array palette (see *Figure 26*).

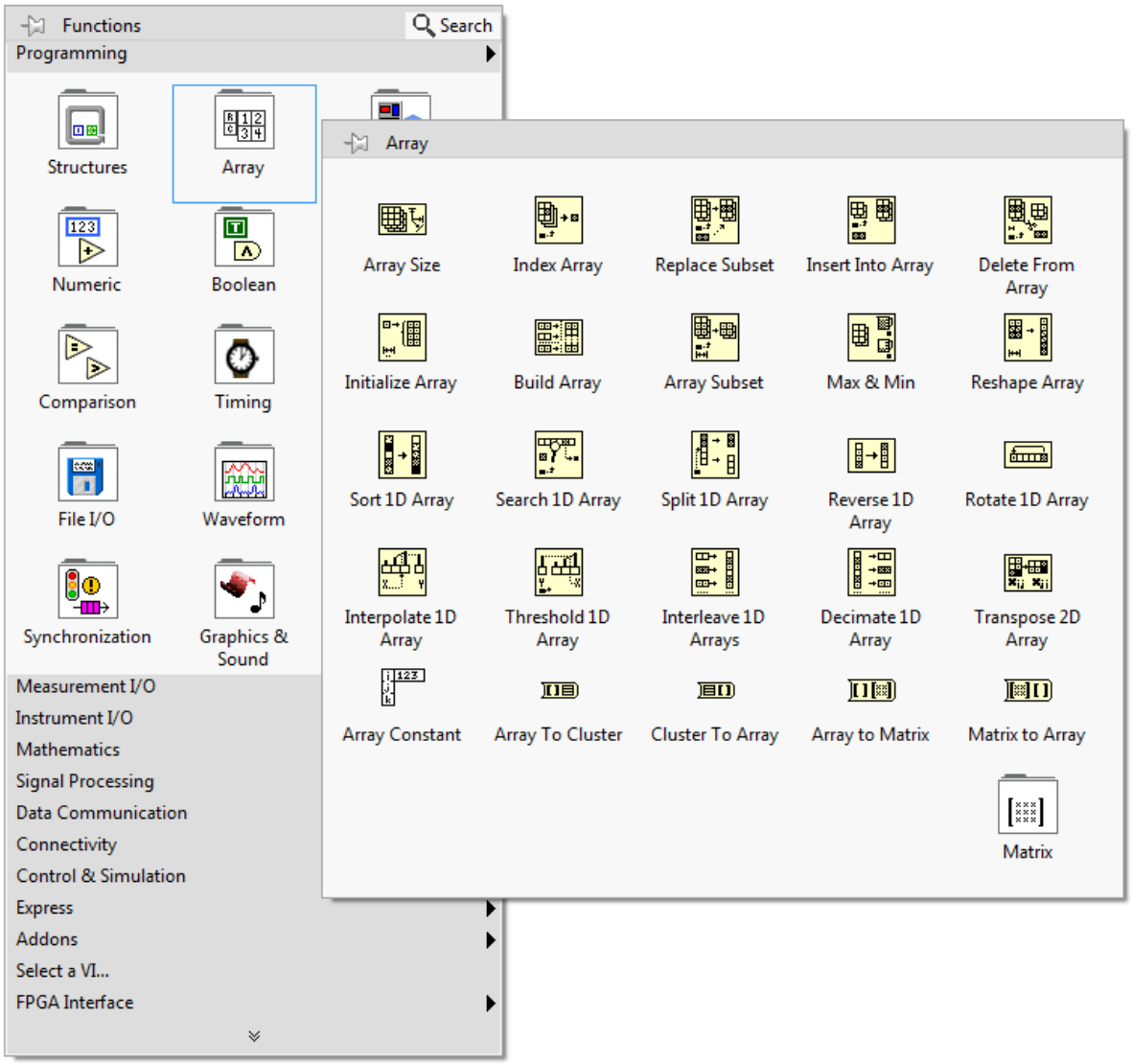


Figure 26: *Array palette.*

The first part of the VI (on the left outside of the while loop) should create the initial conditions of the calculation. Create a 1D array with the 2 first terms of the sequence (see *Figure 27*).

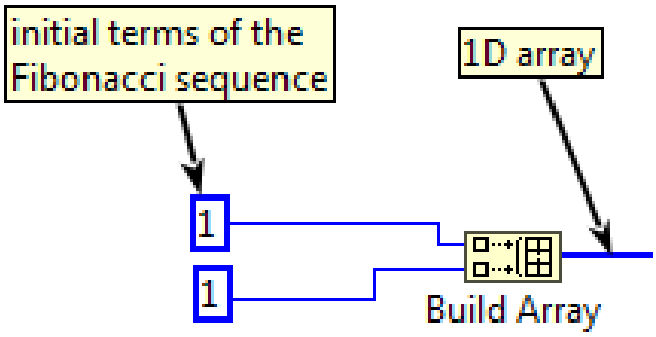


Figure 27: *Initial conditions.*

The 1D array is then passed inside the while loop for performing the iterative calculation. The 1D array is the list of the terms of the sequence. When one calculation iteration is done, the new term (result) is stored in this 1D array (list). Use the "shift register" to pass this 1D array to the next iteration inside the while loop (see *Figure 28*). Add a shift register by right-clicking on the entrance point of the 1D array inside the while loop and select "replace with shift register".

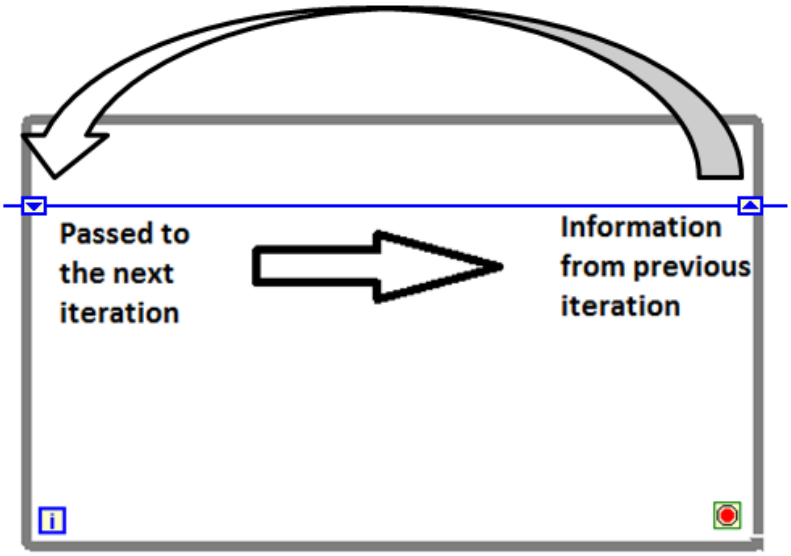


Figure 28: *While loop with shift register.*

Inside the while loop, the iterative calculation has to be performed. Use the "index" function from the array palette to get the first (index = 0) and the second (index = 1) element of the 1D array. Add these two elements to get the new term of the sequence. Store the result at the beginning the 1D array (in this way you always have to get the first and second element of your 1D array to calculate your new term instead of

getting the last and second to last elements). In order to add a new number at the beginning of the 1D array, use the “build array” function with the first input (top) as the new term which has been calculated and the second input (bottom) as the “old” 1D array containing the previous terms of the sequence (see *Figure 29*).

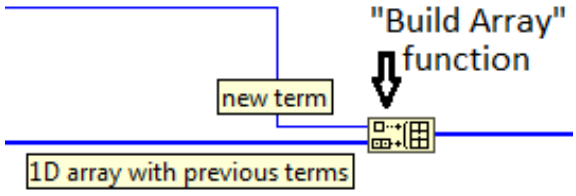


Figure 29: *Adding a new numerical element to an existing 1D array of numerical elements.*

Use the function of the “comparison” palette to check when the result gets larger than the limit number and therefore generate a “true” Boolean signal to stop the while loop.

Once the calculation is finished (and the while loop is stopped), get the 1D array containing all the terms of the sequence and sum them up together. Calculate the number of terms in the sequence by measuring the length of the 1D array with the “array size” function of the array palette. To get the last calculated value of the sequence, get the first element of the 1D array results after it exits the while loop. Convert the numerical results into text string data with the “number to decimal string” function from the conversion palette (see *Figure 30*).

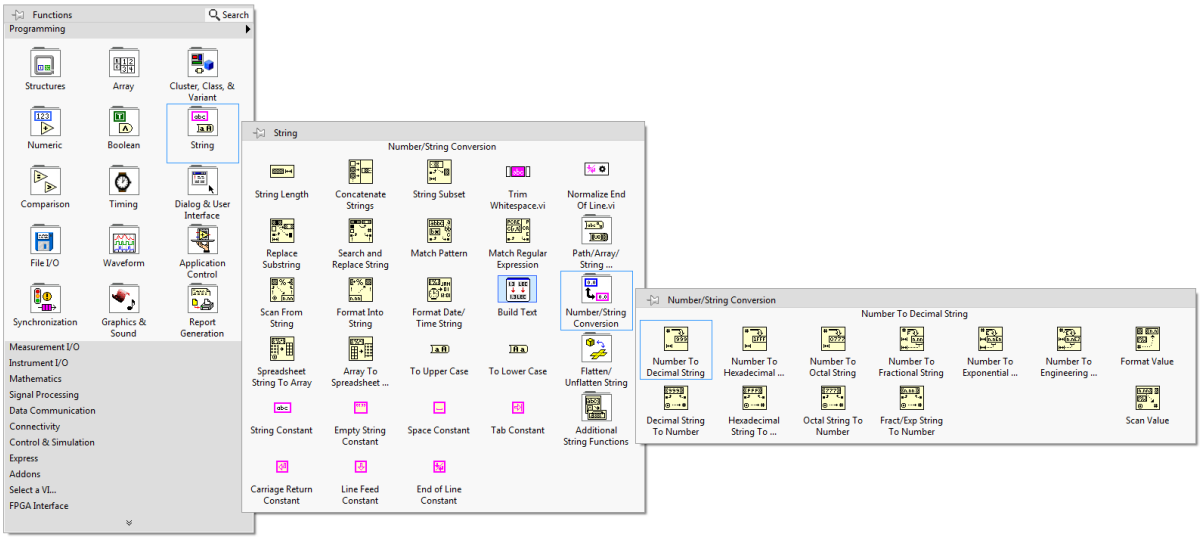


Figure 30: *Number to decimal string function.*

Use the “concatenate string” function and “text string” constants to write down a text presenting the results and display it on the front panel with a text string indicator.

One can add different indicators such as gauge-like meters and speedometer-like indicators by right clicking on the front panel and selecting the numeric palette from the “modern” library (see *Figure 31*). Other styles of numerical and string indicator or control are available in the other style libraries such as “silver”, “system”, “classic” and “express”

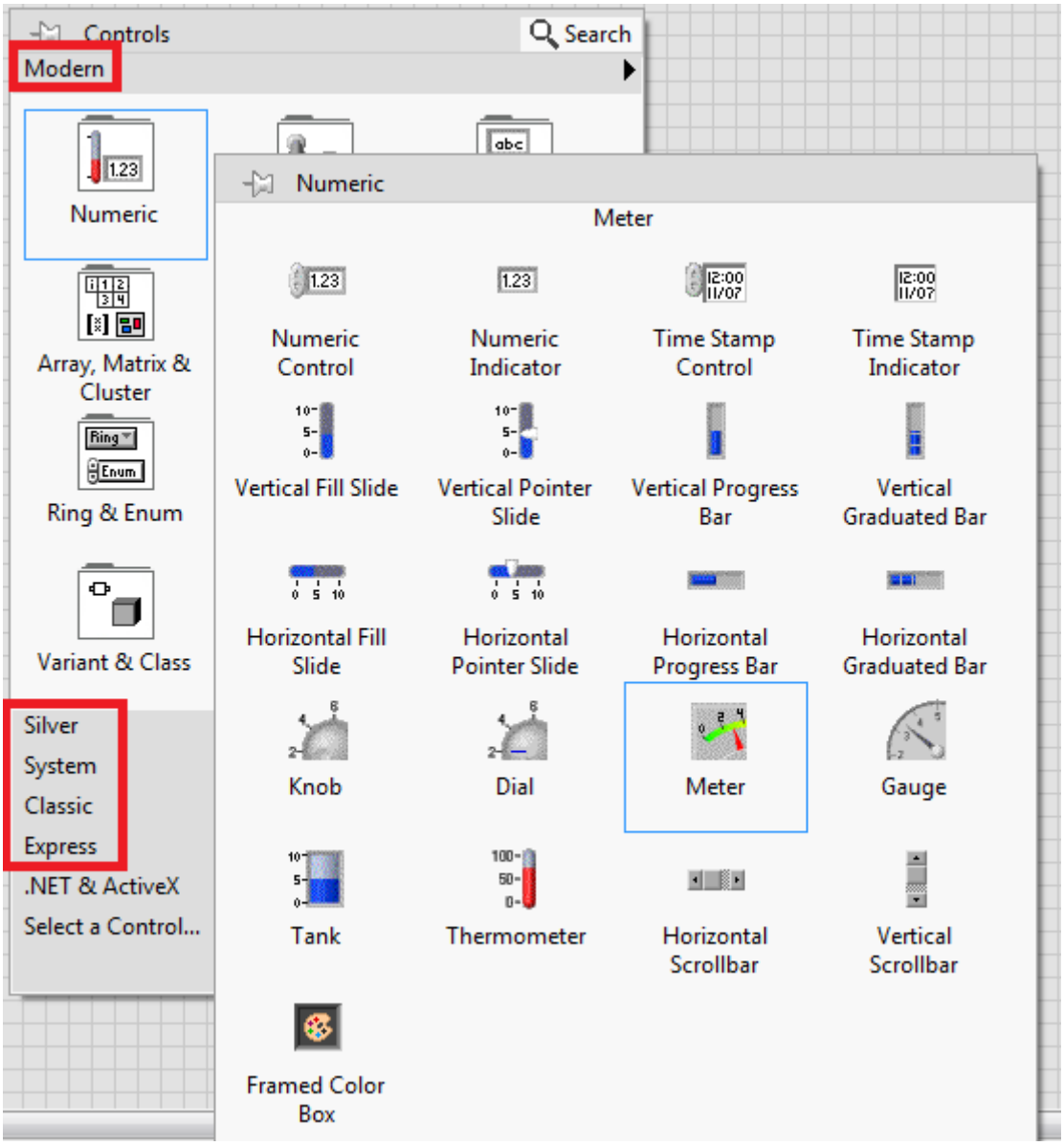


Figure 31: *Adding a speedometer-like indicator on the front panel.*

The different objects of the front panel can be manipulated and edited with the “Tools Palette”. Click on View -> Tools Palette to make the Tools Palette visible / invisible (see *Figure 32*).

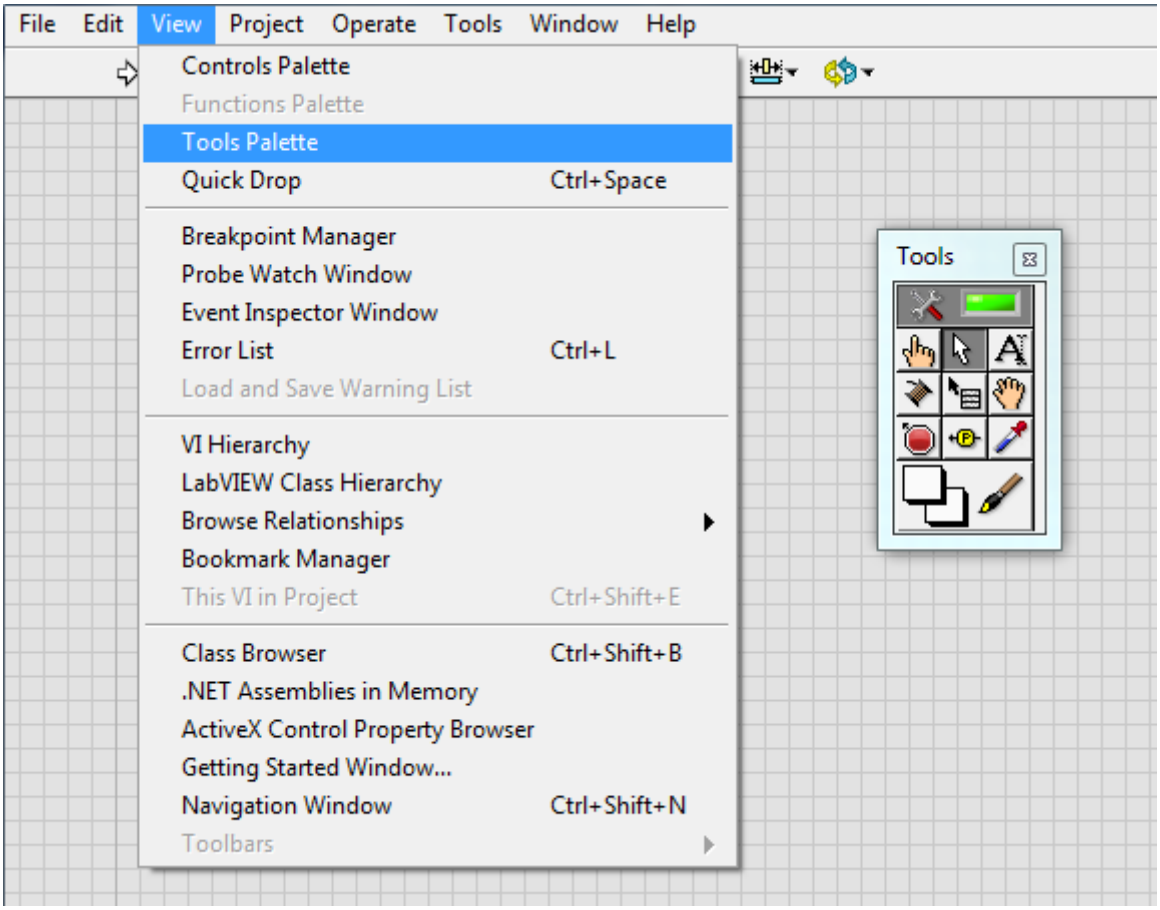


Figure 32: *Make the Tools Palette visible / invisible.*

All elements on the front panel can be reshaped. Rick click -> Properties on a front panel element to access and edit its properties, including appearance, format display and colours. The colour of indicators and other front panel elements can also be changed with the “Set colour” tool (see *Figure 33*).

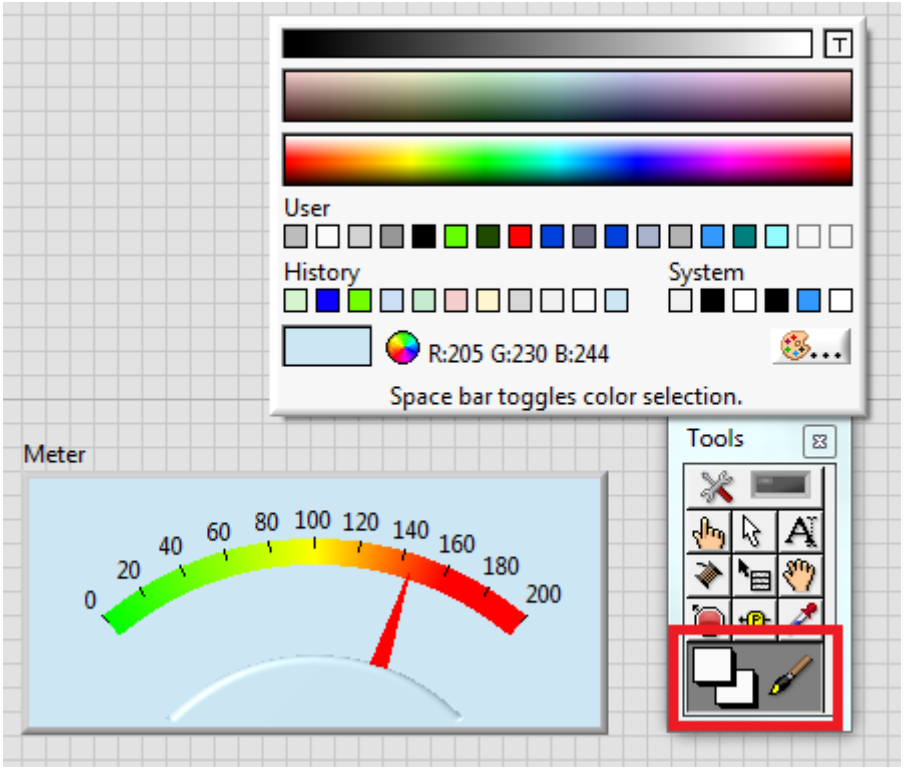


Figure 33: *Set colour of a front panel element.*

Use the “Automatic Tool Selection” for normal operations (see Figure 34)

Figure 34: *Automatic tool selection.*

One can see a summary of all the necessary function for this task on the *Figure 35*.

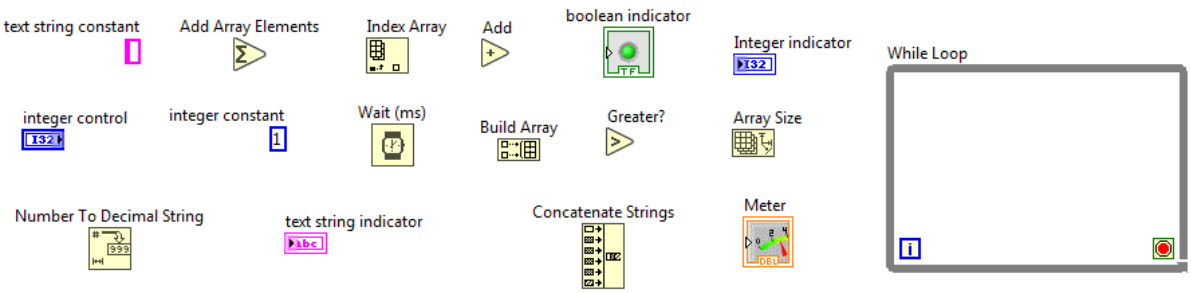


Figure 35: *Summary of all the functions you need.*

Exercise 3

The goal of this third exercise is to build up a VI by using existing sub-functions (sub-VIs) in order to acquire an analog temperature signal from a thermocouple connected to a signal amplifier (see *Figure 36 (**a**)*) through a NI USB-6009 Data Acquisition Device (see *Figure 36 (**b**)*).

Figure 36: *Equipment needed for the exercise 3: Thermocouple connected to signal amplifier (**a**), NI USB-6009 DAQ (**b**), and connection cables.*

First of all, connect the power supply of the thermocouple amplifier and plug the output pins of the amplifier to the analogue input channel 0 (AI0) of the NI USB-6009 (see *Figure 37*).

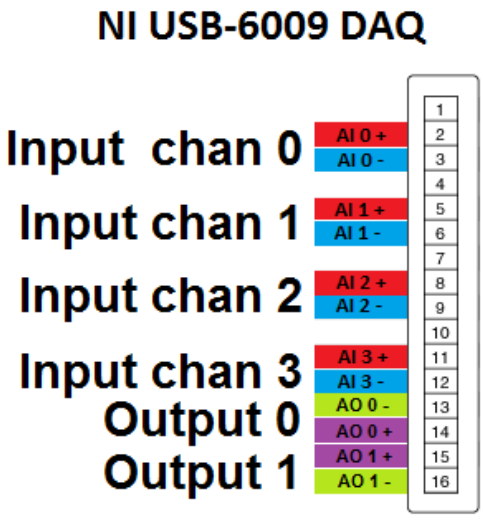


Figure 37: *NI USB-6009 DAQ device pinouts.*

Open the LabVIEW VI shortcut on the computer desktop named “Exercise 3” to open the front panel of the exercise 3 (see *Figure 38*).

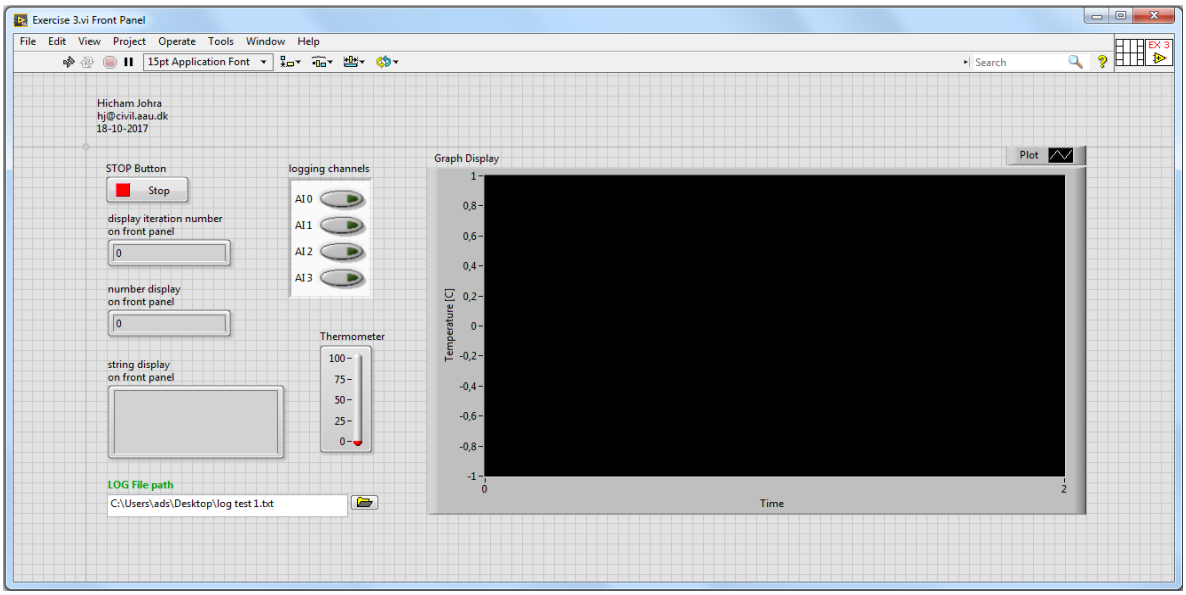


Figure 38: *VI front panel.*

Press “Ctrl + E” to access the block diagram of the VI (see *Figure 39*).

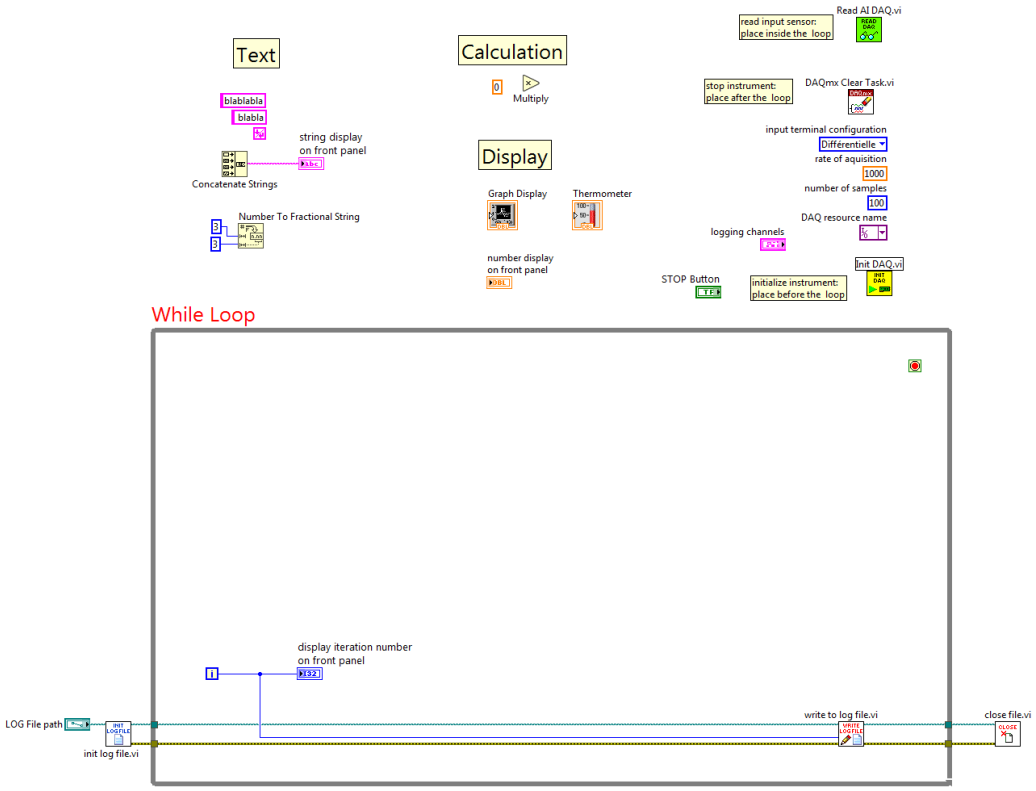


Figure 39: *VI block diagram.*

One can see on *Figure 39* that the block diagram already has some elements (block functions) in place but most of them are not connected and are not at the right position (inside or outside of the while loop). The logging part (creating a log file and writing the data at each iteration) has already been implemented correctly. One can see one *Figure 39* that the log file is created before the while loop is executed (left side of the loop) and closed after the loop is finished executing (right side of the loop). Inside the loop, the dedicated sub-VI is writing the numerical data into the log file at each iteration.

The VI cannot be run (see *Figure 40*) because there are errors in the block diagram. Some input to the block functions are missing. You can click on the broken "run" arrow to get the list of the errors which prevent the VI to be run.

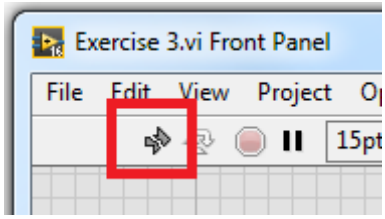


Figure 40: *The VI cannot be run with errors.*

To be able to run the VI, it is necessary to place and connect the different blocks correctly. The general architecture of the VI should **resemble** the one of *Figure 41*.

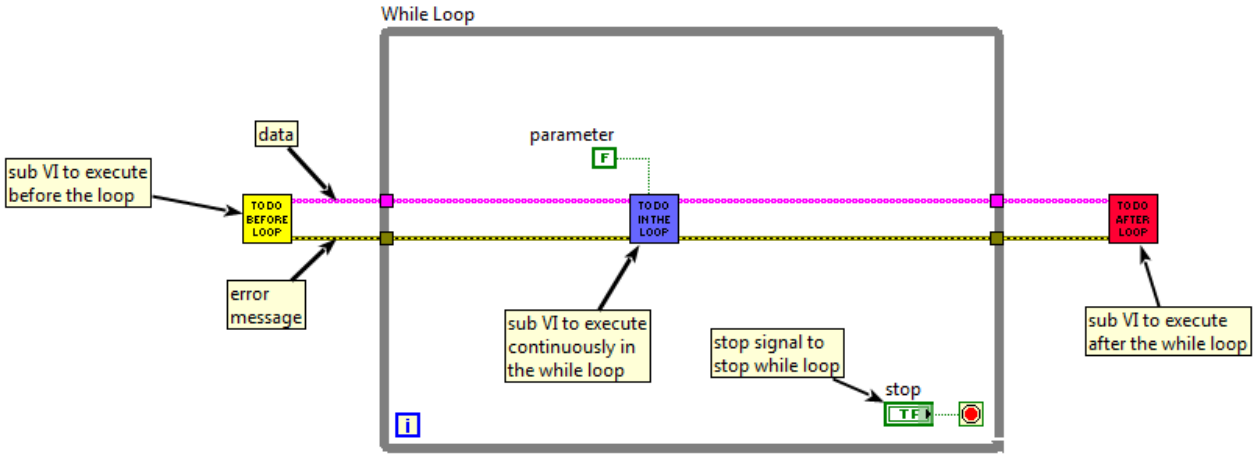


Figure 41: *Continuous execution of a sub-VI inside a while loop.*

The while loop is repeating a piece of code over and over again: continuous execution while the VI is running. There is one function on the left which is executed before the code inside the loop is repeated. There is another function on the right, which is executed once the loop is stopped. The while loop control terminal is connected to a "Stop" button. When the user is pressing the "Stop" button, a "true" boolean signal is send to the stop condition terminal and the loop stops. The iteration index is counting how many times the loop is executed.

Objective

The VI should be able to initialize the instrument and acquire continuously (repeat the “read DAQ” VI in a loop) the analog signal from the sensor on the channel “AI 0”, display it on the front panel with a text string, plot it on a graph and log it in a text file (these should also be performed at each loop iteration).

When the sensor is read, the output of the read DAQ block function (chan AI 0) is converted (simple calculation) from voltage to temperature (in °C). The voltage signal of the amplifier is multiplied by its sensibility (find it on the device) in order to get Celsius degrees instead of volts.

When the user is pressing the “Stop” button, the while loop should stop (a “true” boolean value should be sent to the stop condition terminal of the while loop). Once the while loop is stopped, the DAQ device should be stopped (task clear).

Select the block functions and their parameters and place them on the right position in the block diagram. Then connect the right inputs to the right outputs. Find the right value for the different constants for the calculation or the “DAQ resource name” (see *Figure 39*).

Once all the blocks are connected correctly, you should be able to run the VI. You can first run the VI in highlight mode (step by step) to check if there is any error and if the data flow is correct.

Check the front panel graph to see if it is measuring some temperature. Place you finger on the thermocouples to check if the sensor is responding correctly (see *Figure 42*).

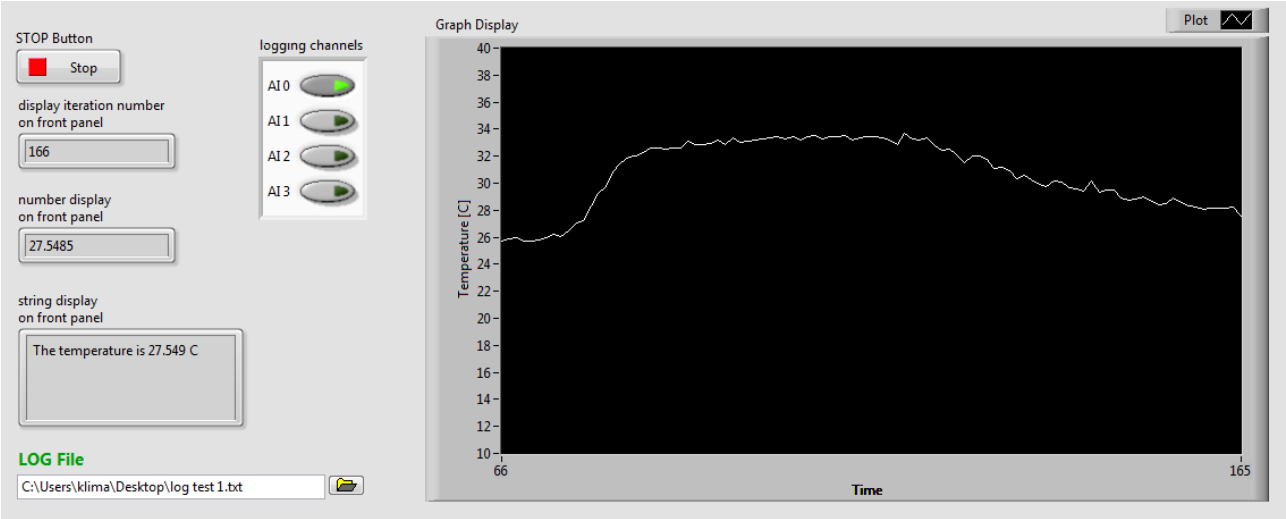


Figure 42: *Correct final result from the VI.*

You can see on the *Figure 42* the temperature signal, the iteration number, and treated data. All data are recorded in a text file (log file). Get the data from this log file and plot it in Excel (copy/paste from the log text file). The graph from the Excel sheet should be similar to the one of the VI front panel (see *Figure 43*).

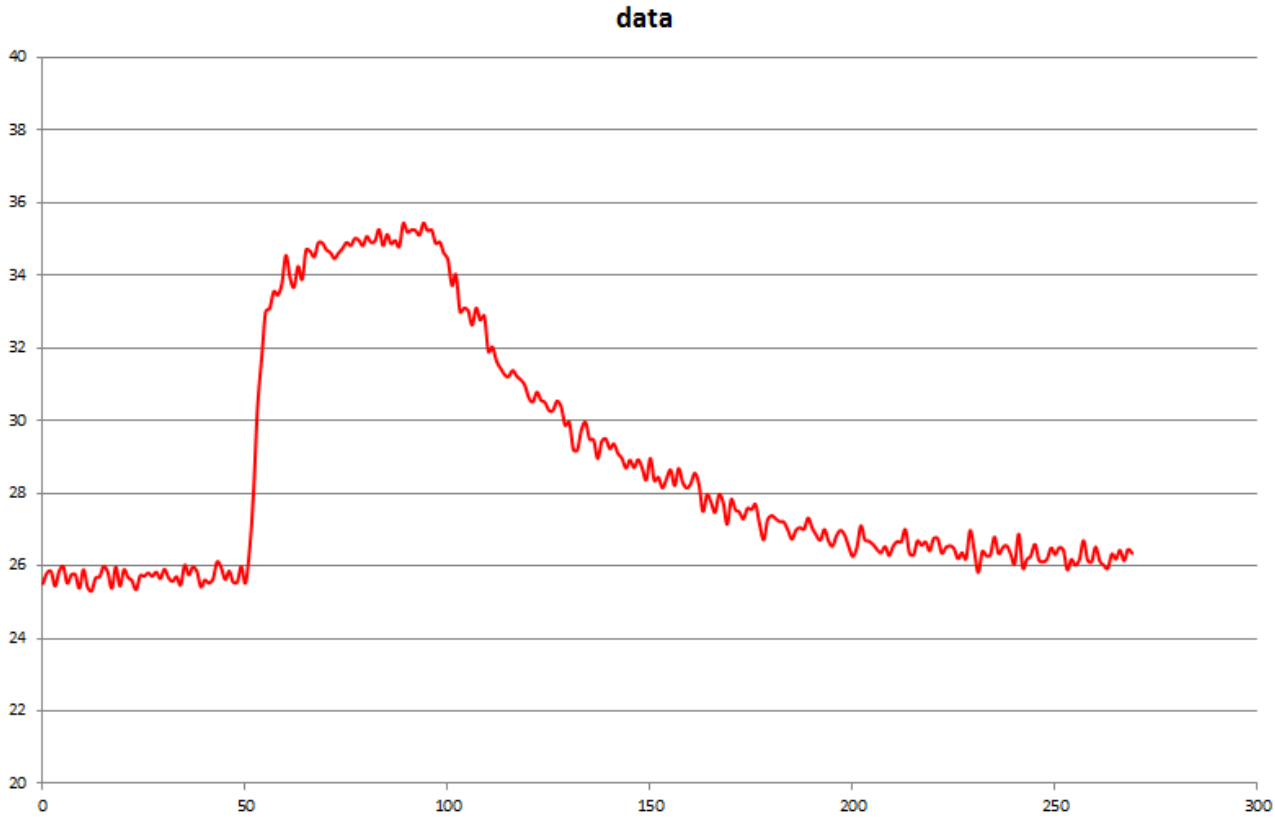


Figure 43: *Excel sheet graph result.*

Help

Do not pay too much attention to the functions themselves or their parameters: they are just doing what it is written they are doing. These functions are themselves sub-diagrams made of other block functions.

Hover over the input and output pins of the block functions to see the names of the inputs and outputs. The input of one block is often the output of another block function or a constant which has the same or similar or related name. It will give you some idea of what to connect with what.

Remember that each type of data has a specific colour. You can only connect the input of a function with the output of another function or a constant which has the same data type (same wire colour).

Remember to insert the right value for the constants of the calculation part and the “DAQ resource name”. Also remember to select the right “LOG File path” on the front panel (select a file which exists) and activate the “AI 0” logging button on the front panel to measure the corresponding channel.

Exercise 4

The goal of this fourth exercise is to add a simple ON/OFF controller for turning on and off an alarm light bulb and a ventilation fan when the temperature measured by the thermocouple is above a certain limit (see *Figure 44*).

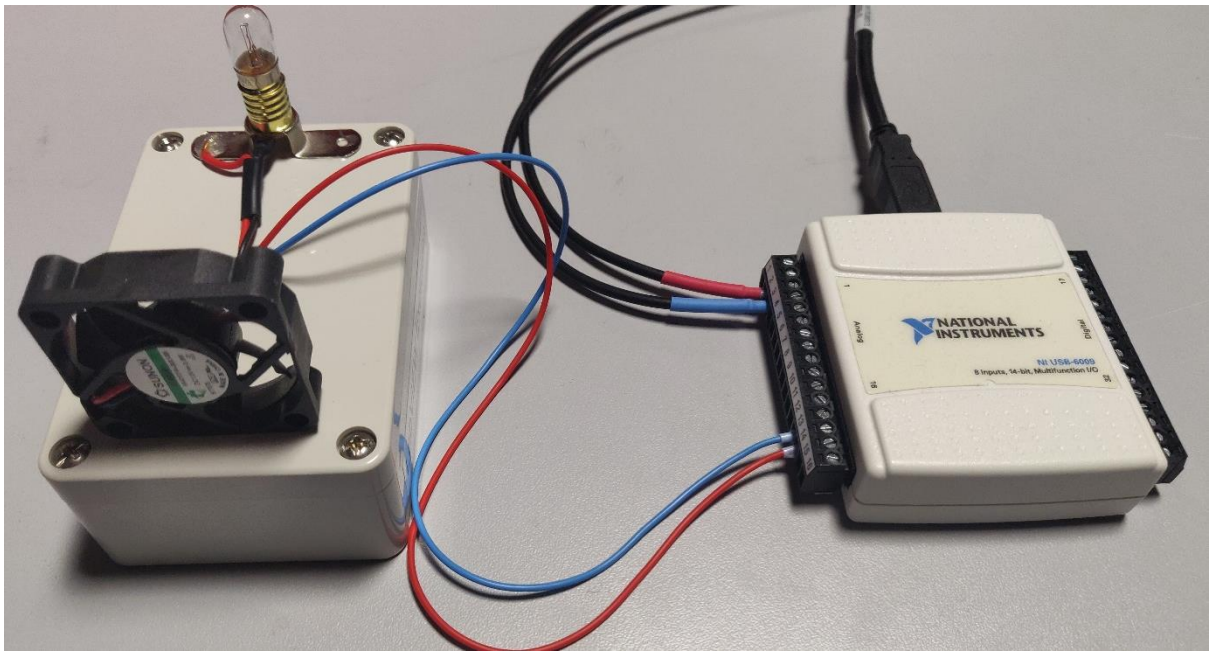


Figure 44: *NI USB-6009 DAQ connected to a controlled box with a light bulb and an electrical fan.*

In the previous exercise, the NI USB-6009 DAQ was used to acquire an analog input signal. This DAQ can also send analog and digital output signal. In the current case, the **red wire (positive +)** and the **blue wire (negative -)**, which control the switch of the light bulb and the electrical fan, should be connected to "Output 0" channel (**AO0+** and **AO0-**) of the NI USB-6009 DAQ (see *Figure 45*).

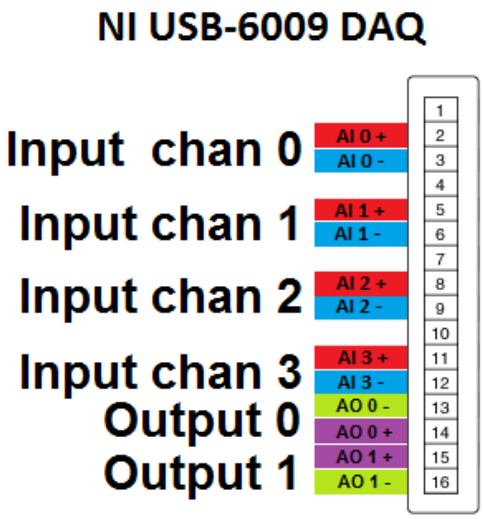


Figure 45: *NI USB-6009 DAQ device pinouts.*

In the previous exercise, the NI USB-6009 DAQ was only used to acquire analog input signal. Therefore, the “Init DAQ.vi” sub-function was only initializing the DAQ for reading analog input signal. In the current case, the “Init DAQ.vi” sub-function has to be replaced by the “Init DAQ 2.vi” which initializes the DAQ for analog input acquisition and for sending analog output signal as well.

To add the “Init DAQ 2.vi” to the block diagram, you can simply drag and drop the file from the “Exercise 4” folder, or “right click -> Select a VI…” The necessary sub-functions for this exercise can be found in the folder “Exercise 4” on the computer’s desktop.

The input parameters of the “Init DAQ 2.vi” function are exactly the same as the “Init DAQ.vi”. However, the “Init DAQ 2.vi” has 2 “task out” outputs. One specific “task out” should be wired to the “Read AI DAQ.vi” sub-function inside the loop to read the temperature input signal (which is what is done in the exercise 3), and the other corresponding “task out” should be wired to the “Write AO DAQ.vi” sub-function. The “Write AO DAQ.vi” sub-function should be added to the block diagram in the same way as the “Init DAQ 2.vi” sub-function. The “Write AO DAQ.vi” is the sub-function which set the DAQ to send a certain voltage analog output (between 0 and 5 Volt).

A “vertical pointer slide” can be added to the front panel in order to allow the user to choose the value of the limit temperature (see *Figure 46*). The control pointer slide can be found in “Silver-> Numeric” palette by right-click on the front panel.

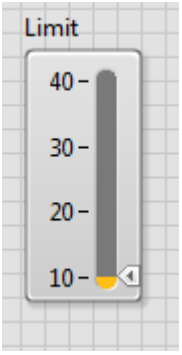


Figure 46: *Control vertical pointer slide from the silver palette.*

On the block diagram, use the functions from the “comparison” palette (see *Figure 47*) to build a simple ON/OFF controller which checks if the measured temperature is higher than the limit temperature set by the control vertical pointer slide. Connect a Boolean indicator to the result of this comparison in order to display on the front panel if the temperature is exceeding the limit or not (a green LED indicator on the front panel is a common way to do so).

If the measured temperature is higher than the limit temperature, 5 Volt should be sent by the DAQ to the switch of the light bulb and electrical fan. Therefore, “5” should be sent as input to the “Write AO DAQ.vi” sub-function placed inside the loop. If the measured temperature is lower than the limit temperature, 0 Volt should be sent by the DAQ to the switch of the light bulb and electrical fan. Therefore, “0” should be sent as input to the “Write AO DAQ.vi”. The “select” function from the “comparison” palette can be used to send either 0 or 5 Volt to the sub-function “Write AO DAQ.vi” in function of whether the measured temperature is higher or lower than the temperature limit.

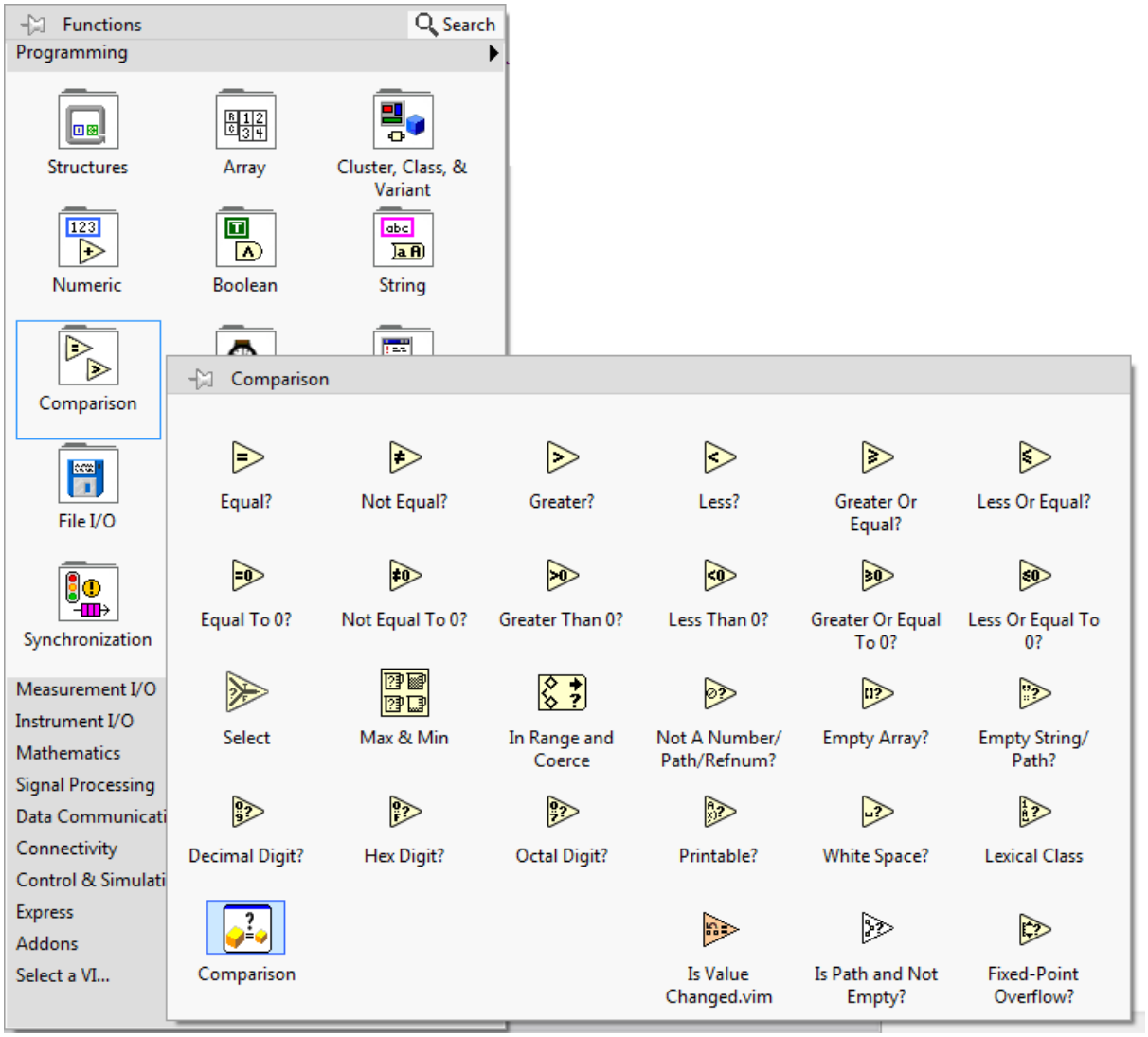


Figure 47: *Comparison palette.*

Run the VI, set the limit temperature a little bit above the currently measured temperature. Warm up the thermocouple to increase the temperature above the limit temperature and check that the controller works: when the temperature is above the limit temperature, the green LED of the VI front panel, the light bulb and the electrical fan are all turned on. They are all turned off when the measured temperature drops below the limit temperature.

If it works: **CONGRATS !!!** You have built your first measurement and control system.

Exercise 5

In the previous exercises, a *National Instruments* equipment was controlled to measure temperature and send back a control signal. Because LabVIEW is developed by *National Instruments*, there are already many simple functions to control all *National Instruments* device. However, it is possible to control any equipment with LabVIEW as long as it has a digital communication port (see *Figure 48*).

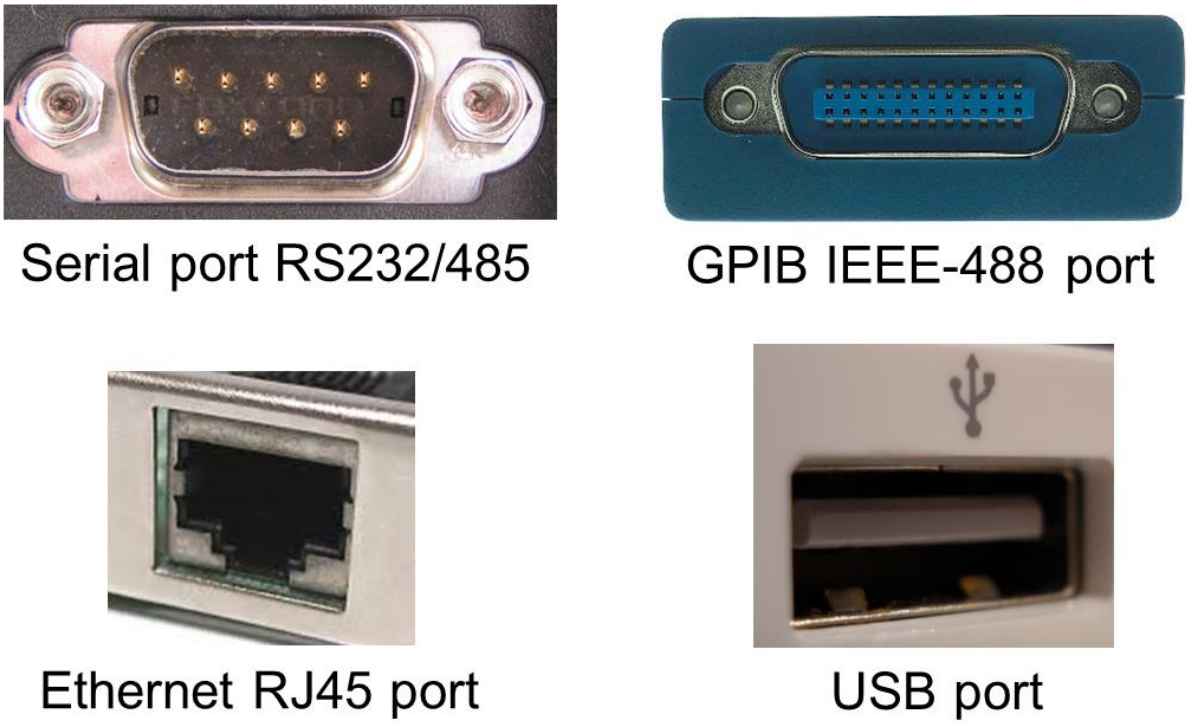


Figure 48: *Different types of digital communication port of devices.*

In order to communicate with these devices, control commands have to be sent according to a specific communication protocol. The communication protocol of a device is specified in its technical documentation. Some communication protocols are very simple and intuitive (usually for expensive devices), consisting of ASCII commands like “reset” to reset the device, or “read chan 1” to read the measurement of the channel 1. On the other hand, some other communication protocols are complicated and not intuitive at all (usually for cheap or industrial purpose devices) and often consist of Hexadecimal commands with very little significance for a human: “Z02” to reset, or “X01” to read measurement on channel 1.

For this exercise, the goal is to send commands to a cheap thermometer device (*Omega iSeries DPi8*) in order to program it to transmit temperature data and change its display colour when the temperature is below or above a certain value (see *Figure 49*). Unfortunately, the communication protocol of this cheap device is not intuitive and some effort has to be made to build correct command messages.

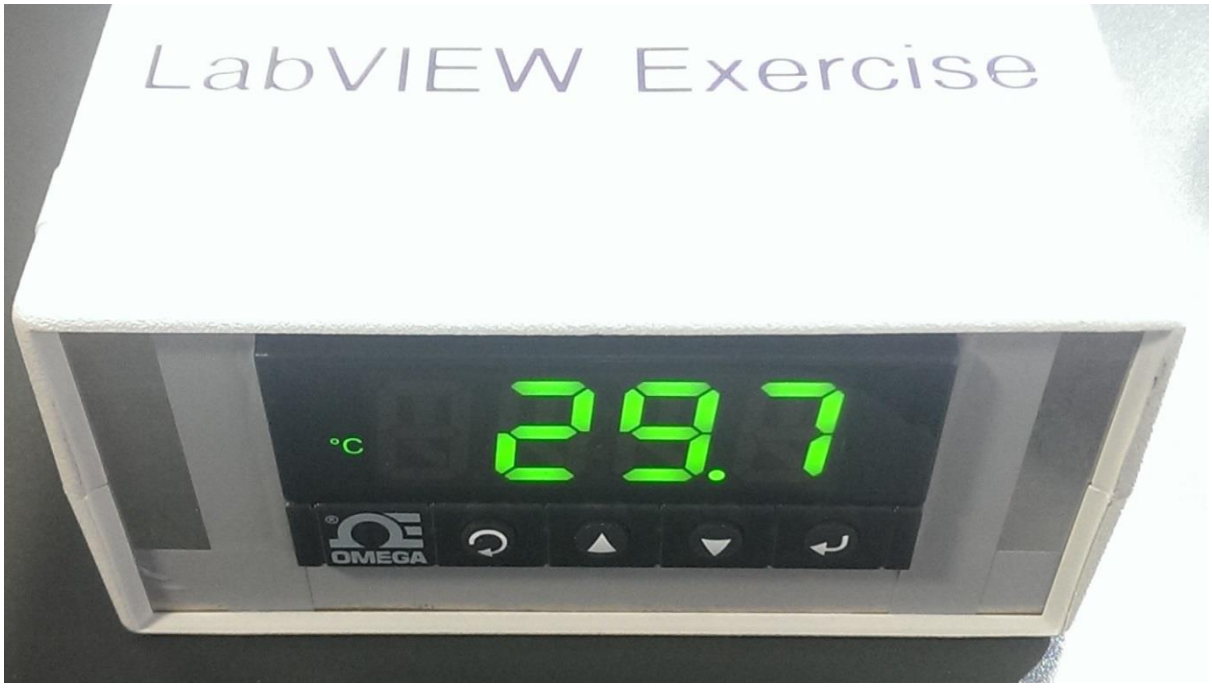


Figure 49: *Omega iSeries DPi8 thermometer.*

VI interface to the thermometer

Open the LabVIEW VI shortcut on the computer desktop named “Exercise 5” to open the front panel of the exercise 5 (see *Figure 50*).

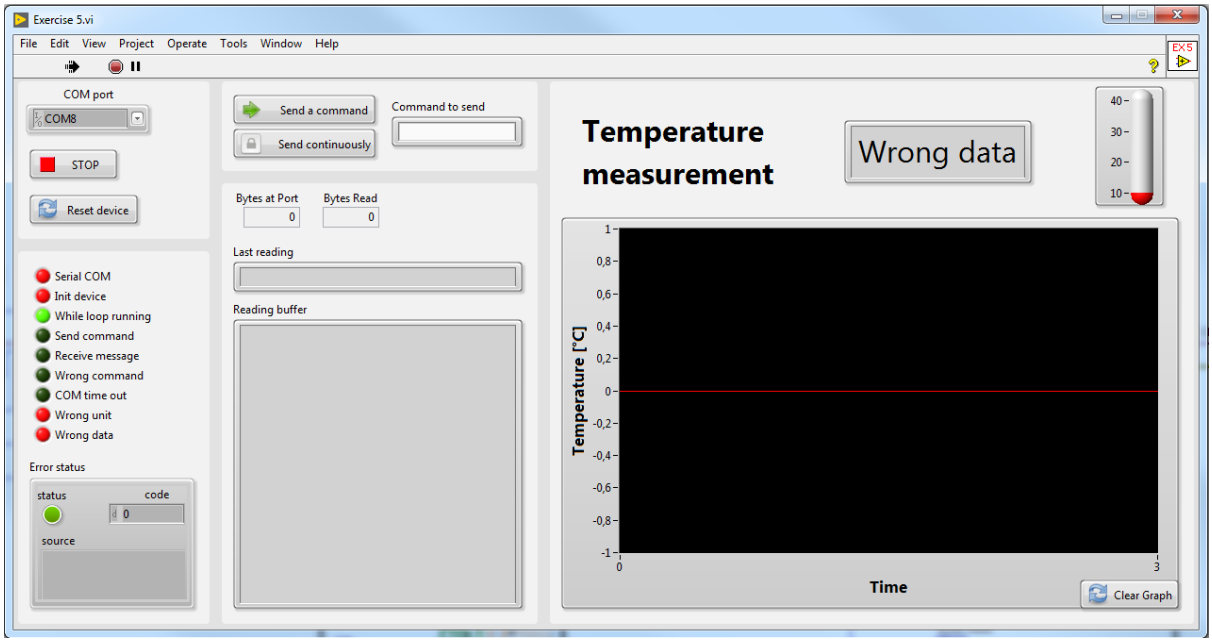


Figure 50: *Front panel of the thermometer interface for exercise 5.*

Open the block diagram of the VI to see the architecture of the interface (see *Figure 51*). One can see that the interface VI is built in a similar way as the previously presented interfaces.

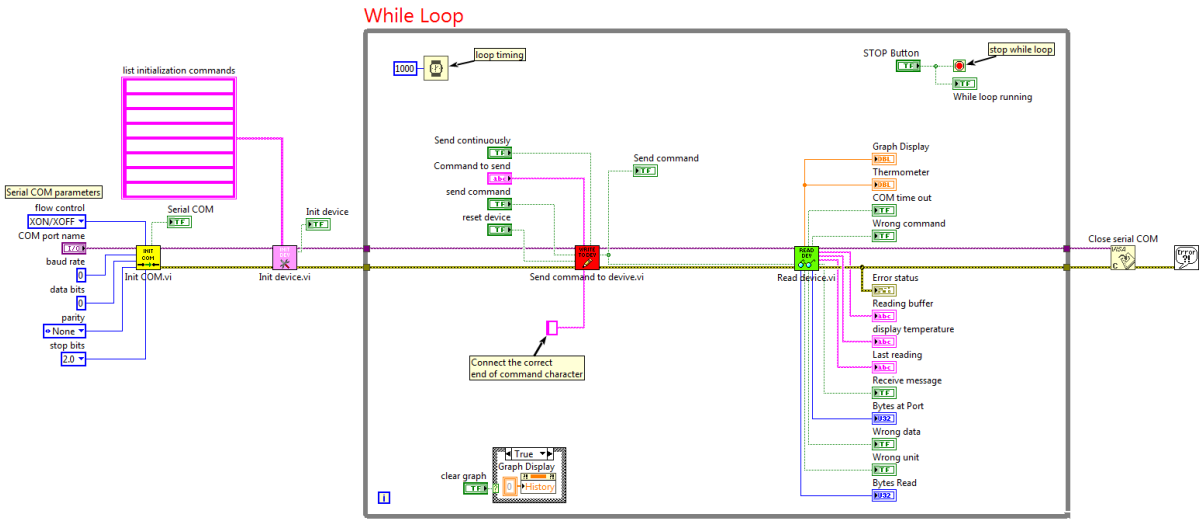


Figure 51: *Block diagram of the thermometer interface for exercise 5.*

There are 2 initialization sub-VIs (“Init COM.vi” and “Init device.vi”) at the beginning of the block diagram (on the left side), before the while loop. The “Init COM.vi” initializes the serial communication between the computer and the thermometer. The inputs to this sub-VI should be the correct serial communication parameters which are specific to the device. If the communication parameters are not correct or if the communication with the device cannot be made, the VI will display an error message.

The “Init device.vi” sends sequentially a list of commands to the device in order to program the latter (data format output, temperature unit, calibration of the probe, alarms, colors…) before performing temperature measurement continuously inside the while loop. The input to this sub-VI is a list of commands to be sent. There is a pause time of 2 seconds in between each command sending.

Inside the while loop, there are 2 sub-VIs: “Send command to device.vi” and “Read device.vi”. The “Send command to device.vi” has different behaviours depending of which button is activated on the VI’s front panel (see *Figure 52*). First of all, make sure that you have selected the correct port name in the “COM port name” controller of the front panel. To send a single command to the device, write the command in the “Command to send” section (**1**) and press the “Send a command” (**2**). In that situation, the command will be send only once to the device, which can be used to test the command and figure out if the command is valid. To send continuously the command (send it once at each while loop iteration, which is once every second), press the “Send continuously” button (**3**). To stop the continuous sending, press the latter again. Press the “Reset device” button (**4**) to send a sequence of commands to the thermometer which will reinitialize it.

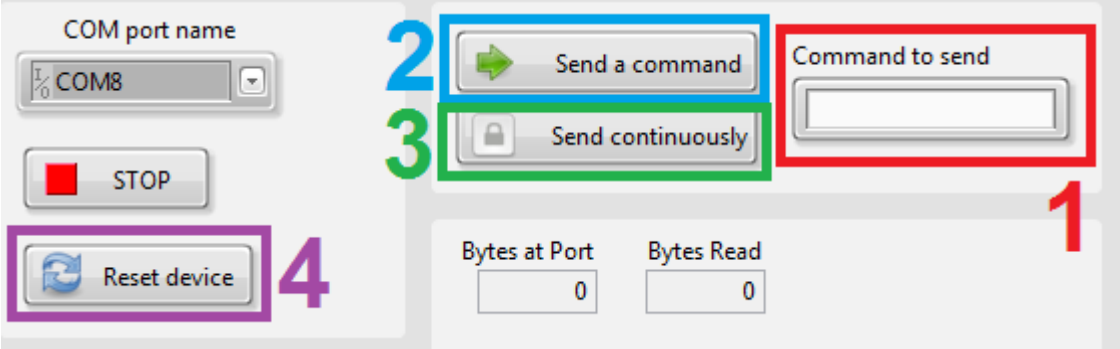


Figure 52: *Front panel elements interacting with the "Send command to device.vi".*

The "Read device.vi" reads and analyses the data coming from the thermometer device. It displays on the front panel how many bytes of information have been received from the device, and shows the last reading (message data) received from the device and the history of the previous message data (see *Figure 53*).

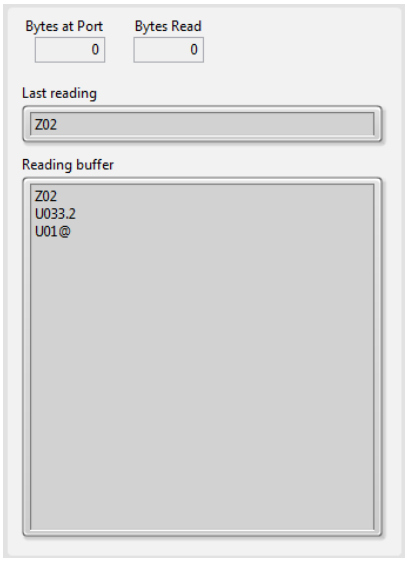


Figure 53: *Front panel elements showing the data response from the device.*

The LEDs on the left side of the front panel (see *Figure 54*) give indications about if the device is working properly or not, and what is the cause of the malfunctioning of the thermometer.

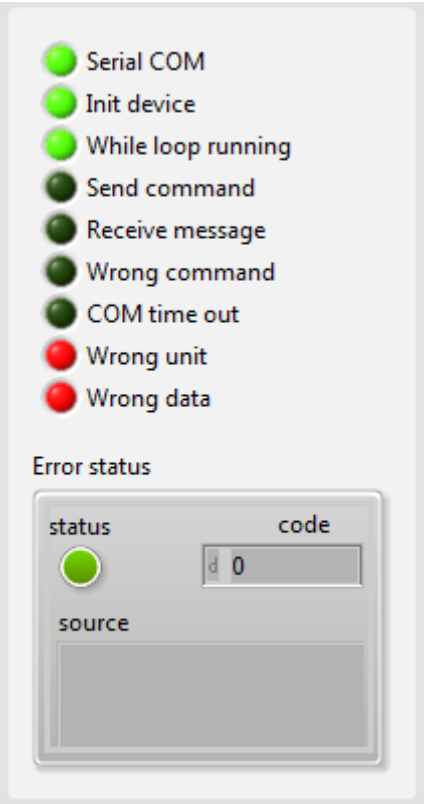


Figure 54: *Front panel elements showing the status of the device.*

The processed data from the device is the current measured temperature. It is displayed on the front panel of the interface and recorded on a graph (see *Figure 55*). Press the “Clear Graph” to reset the graph to zero and erase all recorded temperature data.

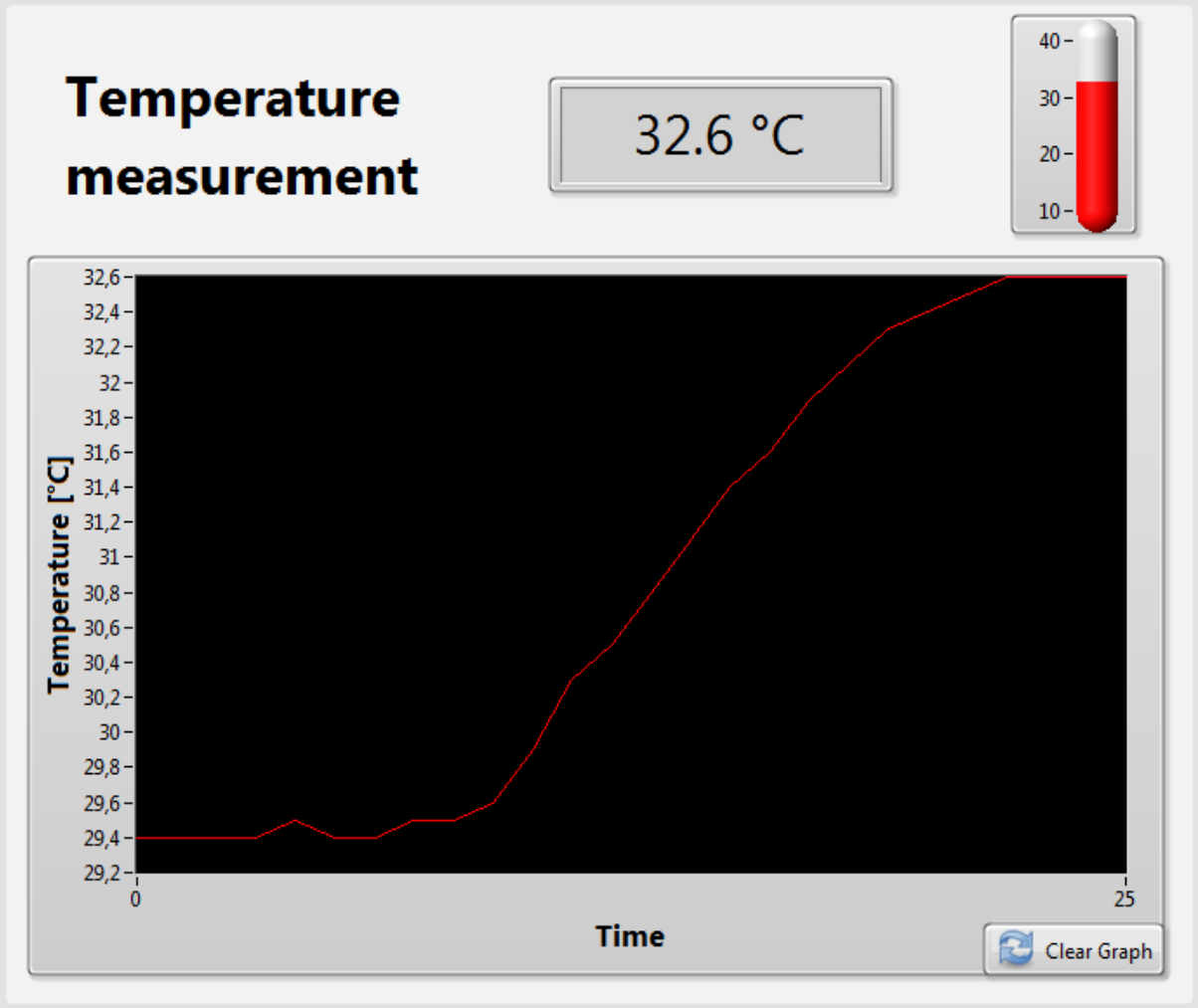


Figure 55: *Front panel elements showing the temperature measured by the device.*

Objective

The goal of this exercise is to find the correct communication parameters and sequence of commands to set the device properly and perform continuous temperature measurement. For that matter, you should find all necessary information in the device's documentation (see "communication manual DPi8 omega thermometer.pdf" file in the "Exercise 5" folder) and test it with the VI interface.

First of all, connect the Omega thermometer to the computer with a serial-to-USB converter (see *Figure 56*).

Figure 56: *Serial-to-USB converter.*

Once the serial-to-USB converter is connected to a USB port of the computer, a new "COM Port" is generated. Before starting the interface VI, selected the right COM port corresponding to the serial-to-USB converter connected to the thermometer (see *Figure 57*). Refresh the list of available COM ports if necessary.

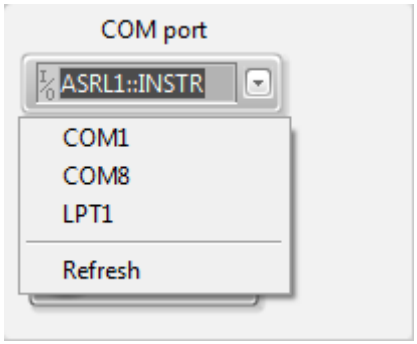


Figure 57: *COM port selection.*

The first task is to find the right serial communication parameters for this device. Inside the block diagram of the thermometer interface VI, the input parameters of the “Init COM.vi” should be changed accordingly (see *Figure 58*). Find the correct parameters in the documentation and test them by running the interface VI.

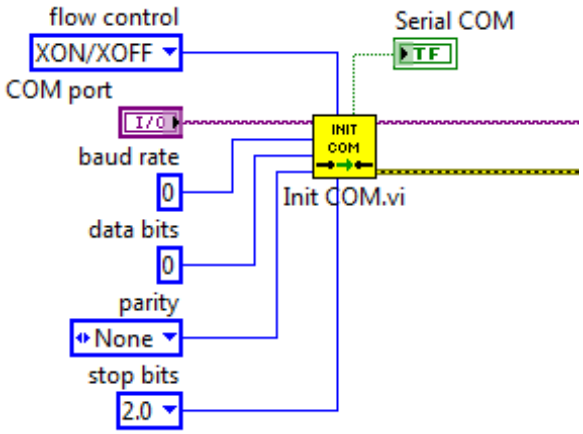


Figure 58: *Serial communication parameters to be changed.*

When running the interface VI, if the serial communication parameters are correct and if the interface manages to establish a communication with the device, the “Serial COM” LED will turn green (see *Figure 59*).

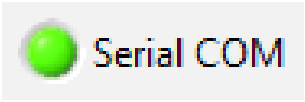


Figure 59: *Serial COM LED turning green when communication has been established.*

Once the serial communication has been properly established with the thermometer, you can start testing the different commands to control the device and initialize it properly. Once the right sequence of commands has been selected, it can be inserted in the vector constant input parameter to the “Init device.vi” (see *Figure 60*).

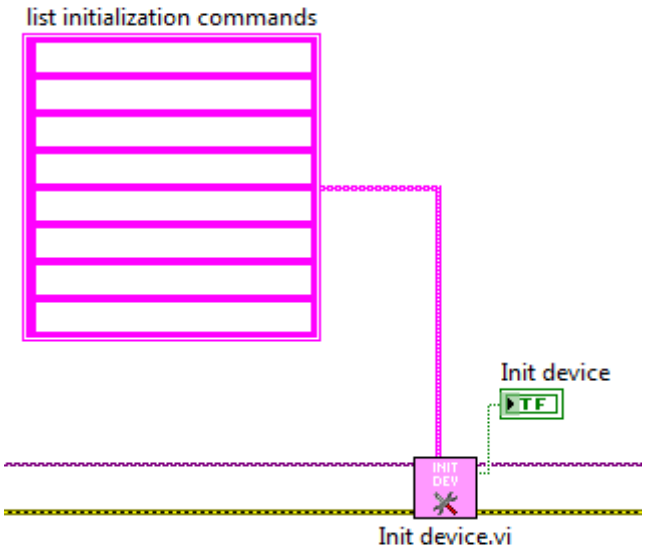


Figure 60: *“Init device.vi”.*

In order to send a valid command message to the device, it is important to end this message with an “end of command” character. Find it in the documentation and replace it as an input to the ”Send command to device.vi” (see *Figure 61*).

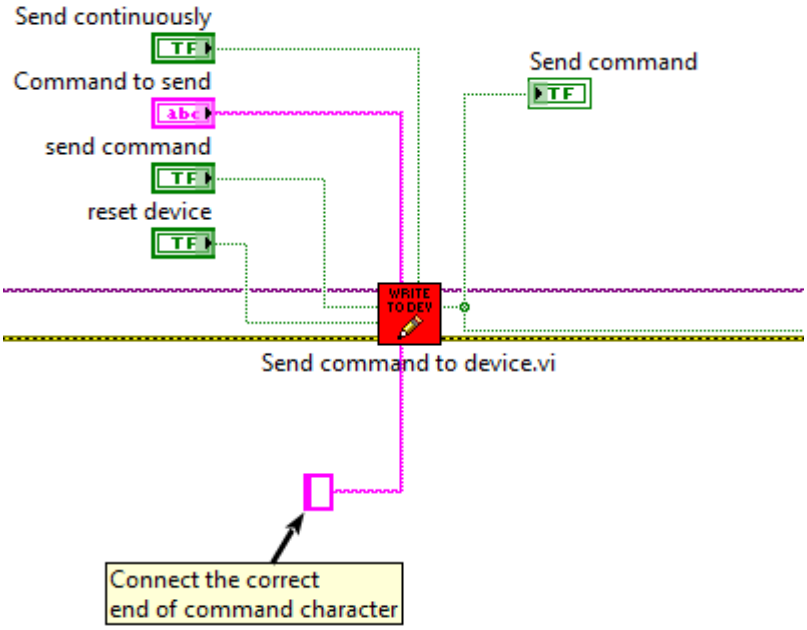


Figure 61: *“Send command to device.vi”.*

Once the correct serial communication parameters and the “end of command” character have been found, the interface VI should be run in order to test the command messages. The goal of this exercise is to find 7 commands which can set the thermometer device in the following way:

- Set the input type format for temperature instrument as 100 ohm RTD with a 392.4 curve.
- Set the reading configuration format for temperature instrument with a filter constant 1, reading in C degrees, and decimal point 2 (FFF.F).
- Set colour display as RED colour for the alarm 1 and for the alarm 2, and green colour for the normal colour.
- Set the alarm 1 configuration as alarm 1 power on enable, loop break time disable, active High/Low, normally open, unlatch, absolute, enable alarm 1.
- Set the alarm 1 low limit to 20 °C.
- Set the alarm 1 high limit to 30 °C.
- Set the data format as unit ON, no valley, no peak, reading ON, no alarm status.

Once the initialization command sequence has been established, find the command which set the thermometer to send one single measurement reading of the temperature as a data string with decimal format. Place this command in the “Command to send” section and run the interface in continuous mode (Press the “Send continuously” button) to continuously ask the device to send a temperature measurement every second. The result of this reading will be automatically recognized and treated by the “Read device.vi” and the temperature measurement will be displayed on the front panel.

Help

You have a pdf file of the documentation: use Ctrl + F to find information faster.

Choose all “Factory Default” values for the serial communication protocol parameters of this device.

There is no flow control for the serial communication protocol of this device.

The “end of command” character is a “Carriage Return Constant”.

To validate a “write” command, or a sequence of “write” commands, the “reset” command has to be sent to the device. Therefore, after sending all the different commands to initialize the device, the “reset” command should be sent once to validate all the initialization sequence of commands.

How to create a valid command message:

A valid command message starts with a “Recognition Character” which is specific to the device. Then follows command prefix (see *Figure 62*). In our case, we want to send a command to the device so that it changes its configurations. Therefore we have to use a “Write” (Write HEX data into EEPROM) command prefix.

COMMAND PREFIX (COMMAND CLASS)	MEANING
^AE	Special read, Communication parameters
P (Put)	Write HEX data into RAM
W (Write)	Write HEX data into EEPROM. **1,000,000 writes to EEPROM is guaranteed!**
G (Get)	Read HEX data from RAM
R (Read)	Read HEX data from EEPROM
U	Read status byte
V	Read measurement data string in Decimal format
X	Read measurement data values in Decimal format
D	Disable
E	Enable
Z	Reset

Figure 62: *Command prefix table.*

After the command prefix, comes the command index (2-digit Hexadecimal number ranging from 01 to 45). Each parameter or function has its specific command index which can be found in table in the documentation.

After the command prefix, comes a 2-digit Hexadecimal number corresponding to the value of the new parameter to be sent to the device. Each specific command index has a table (see documentation) to calculate the correct 2-digit Hexadecimal number corresponding to a certain set of parameter.

Example of how to create a valid command message:

If you want to change the settings for the input type format for temperature instrument, you need to select the command prefix **W** (for "write" to device). According to the documentation, the command index of the input type format for temperature instrument is **07**.

In the documentation, there is a table for calculating the parameter value (see *Figure 63*). To set this parameter as 100 ohm RTD with a 392.4 curve, the Binary command data is **0000 1011**, which is equivalent to the Hexadecimal number **09** (see *Figure 64*).

The correct command message to be sent is therefore **W0709** (the recognition character is missing here. It should be added to form a proper command message).

5.7.1 Input Type (Command Index 07)

5.7.1.1 Input Type Format for Temperature/Process Instrument

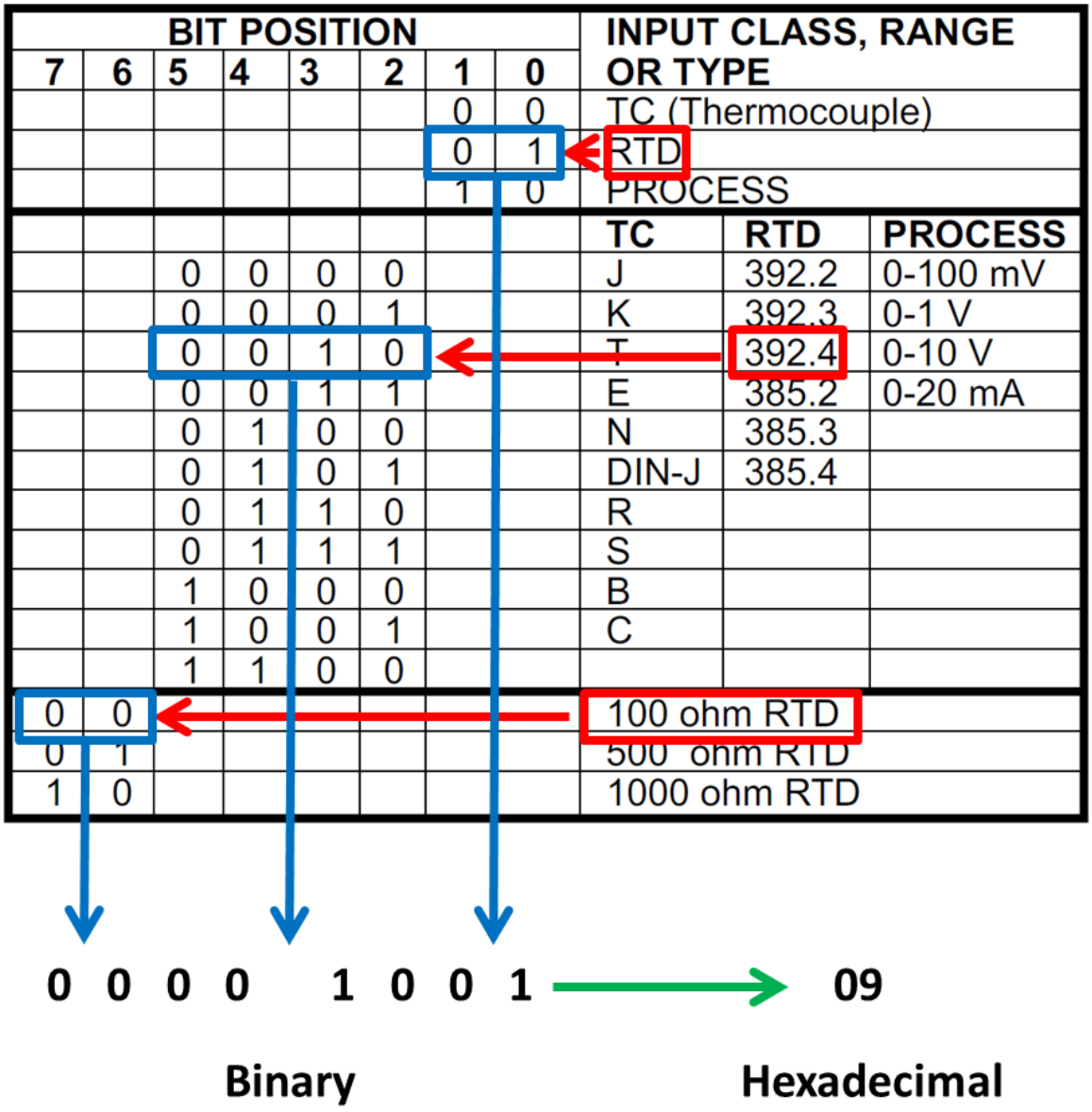


Figure 63: *Table to create a correct Hexadecimal command data for the command index 07.*

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Figure 64: *Binary to Hexadecimal conversion table.*

Additional help for the alarm settings:

The Hexadecimal data command for the low and high alarm limits are more complicated to calculate, therefore the command messages are given hereafter:

- To set the alarm 1 low limit to 20 °C, send the command **W122000C8**.
- To set the alarm 1 high limit to 30 °C, send the command **W1320012C**.